BOEING

(NASA-CR-143892) SSME STRUCTURAL COMPUTER PROGRAM DEVELOPMENT: BOPACE PROGRAMMER MANUAL (Boeing Aerospace Co., Seattle, Wash.) 91 p HC $4.75 CSCL 09B

N75-27776

Unclas

G3/60 29184

THE BOEING COMPANY

D5-17266-4

SSME STRUCTURAL COMPUTER PROGRAM DEVELOPMENT

BOPACE PROGRAMMER MANUAL

CONTRACT NAS8-30615

APRIL 15, 1975

PREPARED BY

BOEING AEROSPACE COMPANY
RESEARCH AND ENGINEERING DIVISION
SEATTLE, WASHINGTON 98124

R. G. Vos - Technical Leader
J. W. Straayer - Program Manager

Prepared For

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
MARSHALL SPACE FLIGHT CENTER, ALABAMA 35812

TABLE OF CONTENTS

SECTION		PAGE
	Table of Contents	ii
	List of Illustrations	iii
1.0	Introduction	I-1
2.0	Main Program Flow Logic	2-1
3.0	Subroutine Definitions	3-1
4.0	Input/Output and Common Storage	4-1
4.1	Input/Output Files	4-1
4.2	Common Blocks	4-2
5.0	Core Reductions	5-1
5.1	Overlay	5-1
5.2	Dynamic Storage of Arrays	5-1
6.0	Linear Equation Solver	6-1
6.1	Introduction	6-1
6.2	User Interface	6-10
6.3	Programming	6-13

LIST OF ILLUSTRATIONS

Figure No.		Page
2.0-1	Program Flow	2-2
5.1-0	BOPACE 2-D Overlay Diagram	5-2
6.1-1	Two Bar Structure	6-2
6.1-2	Three Bar Structure	6-7
6.3-1	Partitioned Matrix Tape Format	6-16
6.3-2	Decomposition Tape Format	6-18
6.3-3	Header and Trailer Records Format	6-20
6.3-4	Nodal Record	6-21
6.3-5	General Flow Generate/Merge Routines	6-22
6.3-6	Decomposition Flow	6-37
6.3-7	Partition Row/Column ID Bookkeeping Array	6-39
3.1-1	BOPACE Isoparametric Ring Element	3-3

1.0 INTRODUCTION

This document is the Programmer Manual for the 2-dimensional BOPACE (Boeing Plastic Analysis Capability for Engines) program. It completes a series of BOPACE 2-D documents, of which the previous volumes were Theoretical Manual, User Manual, and Demonstration Analysis.

BOPACE is written in FORTRAN IV and has been extensively run on both the IBM 360/370 and Univac 1108 computer systems. Two versions of the program are available. The first is a 300-OOF version developed for fast analysis of small size problems within moderate core-storage limitations. The second is the basic 1000-DOF version. In addition, a low-core modification of the 1000-DOF version has been developed (mainly for use within a 64K core restriction on the Univac 1108) through the use of dynamic storage of arrays. Recently a new plastic-creep algorithm has been developed for improving the iterative speed and convergence of BOPACE (see Addendum to BOPACE Theoretical Manual, document D5-17266-1, or the BOPACE 3-D document D180-18677-1). This algorithm has been incorporated into the 300-DOF BOPACE version for further use and evaluation.

2.0 MAIN PROGRAM FLOW LOGIC

This section summarizes the BOPACE flow logic, and discusses key instructions and variables in the MAIN program. Subroutine descriptions and their variable definitions are given in Section 3.

Figure 2.0-1 is the program flowchart, and it is suggested that this figure and a listing of the MAIN program be used in following the discussion given below.

Before step 1 of the flowchart is performed, several preliminary setup instructions are executed. A value is set for NSTOR, defining the size of the array STOR which is used as a core storage area for the system matrix merging and decomposition. The various disk files UNITE1 to UNITS2 are assigned unit numbers (these files are described in Section 4), and a date statement is used to define the group of NMAX values which set various maximum size capabilities of the program. All variables which occur in the MAIN program are explicitly defined in type statements (integer, real, or double precision) in order to give greater visibility to the programmer.

Step 1 of the flowchart is the first step which is executed for each new problem in the data deck. Subroutine READRS is called to read various user-defined file numbers, including input and output restart (checkpoint) tape numbers. READO is then called to read various codes and incremental-iterative control constants.

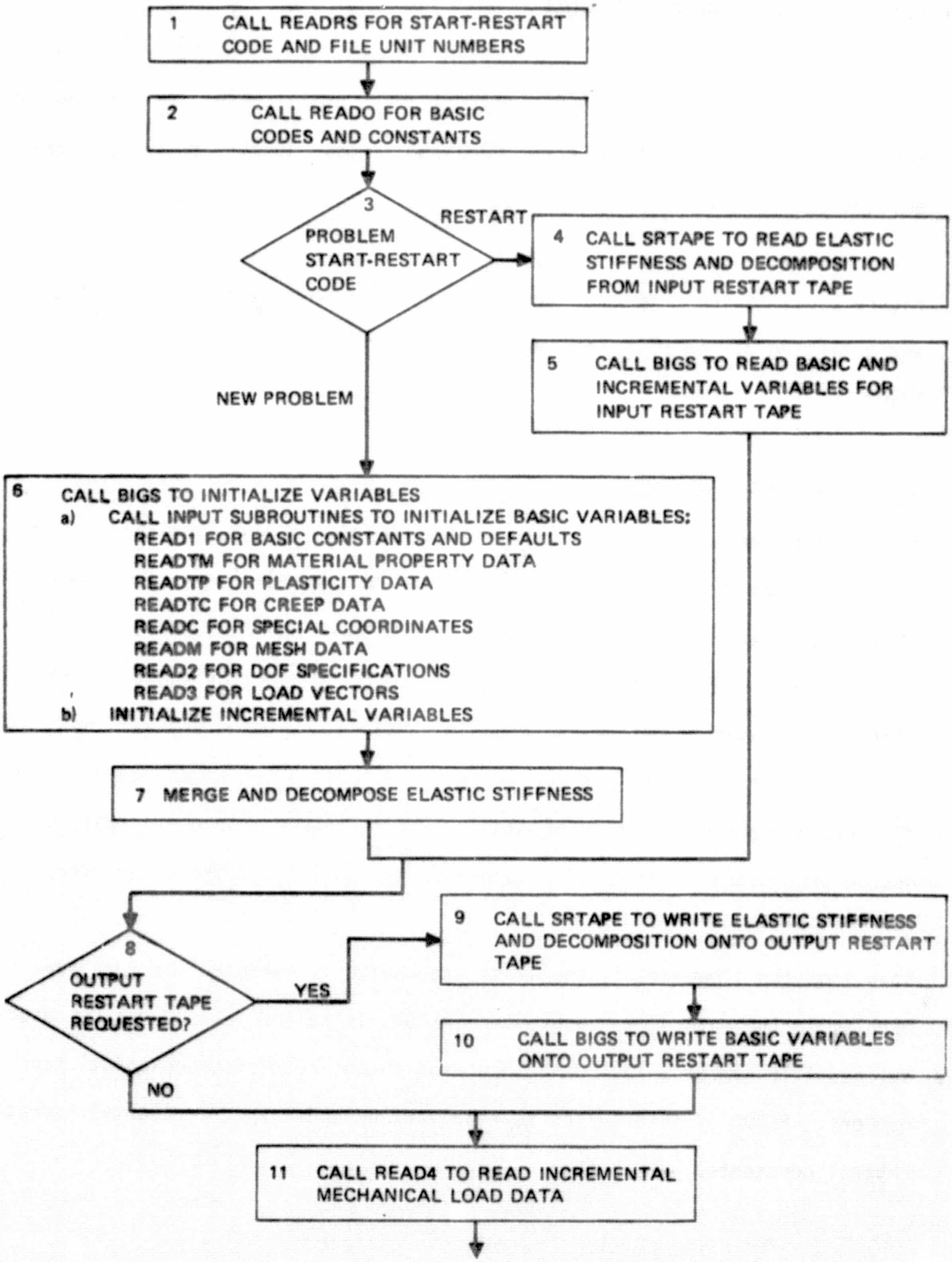


Figure 2.0-1. Program Flow

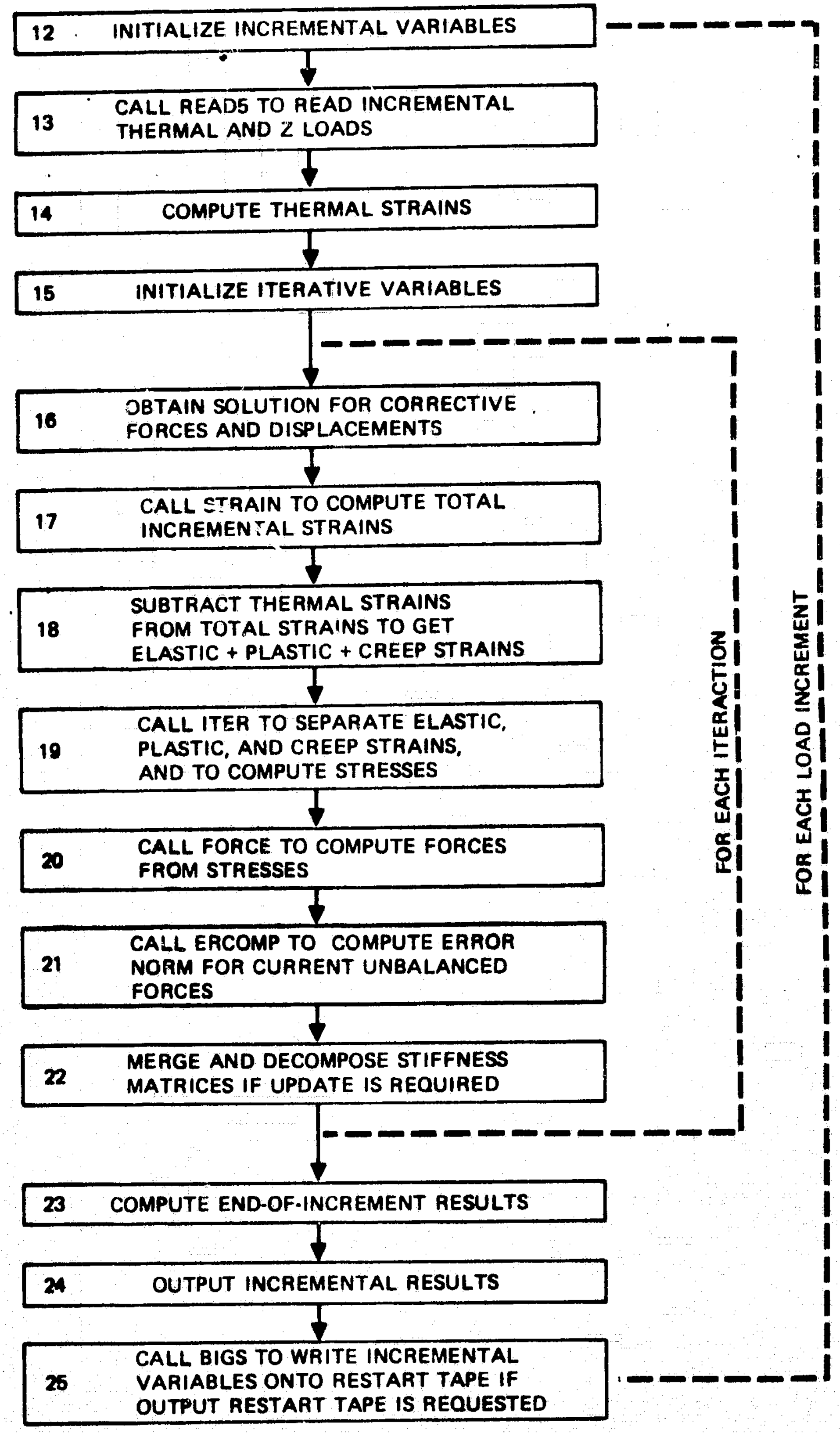


Figure 2.0-1. Program Flow (Continued)

If the input restart tape number (variable UINRS) was not provided by the user, it has a value of 0 at this point, and a cold start (i.e., a new problem not previously run) is indicated. In this case the subroutine BIGS is called in step 6 to serve as a master calling routine for the other input routines and to initialize variables. Since a new problem is being run, its system stiffness matrix (initial elastic matrix) is merged and decomposed in step 7 by the MERGE and DECOMP routines.

If UINRS is nonzero, however, a problem restart is indicated. In this case steps 4 and 5 are executed. Routine SRTAPE is called to read the initial elastic stiffness matrix and its decomposition from the restart tape. The variable DUM (the maximum number of active nodes in the solution wavefront, which is stored in common block JLB) is then read. Routine BIGS is also called to initialize the required basic and incremental variables by reading them from the restart tape.

If the output restart tape number (variable VOUTRS) has been given a nonzero value by the user, problem data must be saved for a possible future restart. Therefore, routines SRTAPE and BIGS are called to write the initial portion of the restart tape. DUM = maximum wavefront nodes is also written on this tape.

Next the value of UNITP, which is the unit number for storage of the total (elastic + plastic) tangent stiffness matrix decomposition, is initialized to that of the elastic matrix decomposition, UNITE2. A value is also set for NN = number of degrees of freedom (including constrained freedoms) in the total system. Routine READ4 is then called in step 11 for all load increments

to be solved for in the current run. This data is read, checked and stored for all load increments at this point, in order to ensure their correctness for later use.

The load increment loop, which is the major outer loop of the MAIN program, is now started. In step 12 various start-of-increment variables (generally denoted by a final 0 in their names) are set to the values of the corresponding end-of-increment variables (generally denoted by a final 1 in their names). This is done by looping over the degrees of freedom (1 to NN) and elements (1 to NEL). P0, P1 are the total applied nodal loads. The P1 vector is computed, using the load factors PFACT and the load reference vectors PREF corresponding to the two input loading distributions. PAR is the incremental loading, equal to P1 - P0. Incremental displacements DQ and internal (resisting) forces DP, are initially set to zero, and the unbalanced (external - internal) forces P are initially set to the incremental loads PAR. T0, T1 are the element temperatures, and Z50, Z51 are the element Z-loads (user prescribed normal stress for a plane stress problem or prescribed normal stress for a plane stress problem). DEC0, DEC1 store the incremental creep strains, and DEP0, DEP1 store the incremental plastic strains. Because the material theory assumes plastic and creep strains to be incompressible (sum of the X, Y, and Z strains is zero), the Z-direction strain can be computed from the X and Y strains and it is therefore not stored. If no creep occurred during the previous increment (this is determined by checking the sum of absolute values of the DEC1 components) then DEC0 is not set to DEC1, because it desired to have DEC0 always store the values from the most recent nonzero creep increment. If no creep is to occur during the present increment (creep

time increment CTIME is zero) the DEC1 is set to zero for the increment. Plastic strains are treated similarly. If an element underwent plastic deformation during the previous increment (i.e., the elastic-plastic condition code YCODE1 > 0, and DEP1 has at least one nonzero component), the DEPO for that element is set to DEP1, so that DEPO always stores the values from the most recent nonzero plastic deformation increment. The condition code YCODEO is set to YCODE1, and since it is not yet known whether an element will be elastic or plastic during the present increment, its YCODE1 code is set to a neutral zero value. YIELDO, YIELD1 define the yield surface size (based on isotropic hardening as a function of plastic deformation and temperature), and are equal to one half the algebraic difference between uniaxial tensile and compressive yield stresses. TWORKO, TWORK1 are the cumulative plastic work densities. TBASEO, TBASE1 and EBASEO, EBASE1 are the plastic hardening parameter values for isotropic and kinematic hardening, respectively. CWORKO, CWORK1 are the cumulative creep work densities. CBASEO, CBASE1 are the creep hardening parameter values. SIGMAO, SIGMA1 are the cumulative stresses, and ALPHAO, ALPHA1 are the cumulative stress-center values. Because increments in stress-center values are proportional to increments in plastic strain, the Z values can again be computed from X and Y values and are therefore not stored.

Routine READ5 is now called in step 13 to read end-of-increment values for temperatures T1 and Z-loads ZS1. Incremental thermal strains DETEMP are then computed in step 14 using the YVA1 table interpolation routine, with the thermal strain tables and temperatures. NUP = number of stiffness matrix updates, and NIE = number of residual-force iterations performed using the

elastic stiffness matrix, are initialized to zero in step 15. The residual error norms ERR0, ERR1 for the previous, current iterations are also initialized, and a new printer output page is readied.

The residual-force iteration loop, which is the major inner loop of the MAIN program, is now started. The first task of the iterative scheme is to obtain a better estimate for the incremental displacements DQ in step 16. Usually this is accomplished by the forward-back-substitution equation solving routine SOLN, which computes displacement corrections Q corresponding to current unbalanced forces P, using either the elastic (UNITE2) or plastic (UNITP) decomposed stiffness matrix file. However, another method for obtaining a better estimate for DQ may be employed, in certain cases if convergence is not occurring (i.e., error norm, ERR1 > ERR0). In such a case the new displacement correction is not computed from the unbalanced forces; rather the previous displacement correction Q is multiplied by some reduction factor CUTF to obtain the new value for Q. In either case the new value of Q is added to DQ, to obtain an improved DQ estimate.

Routein STRAIN is then called in step 17 to compute the incremental strains DE corresponding to displacements DQ. DE is the sum of the incremental thermal + elastic + plastic + creep strains. In step 18 the thermal strains DETEMP are subtracted from DE, and the Z-component of DE is set using the user prescribed increment of normal strain or stress (ZS1 - ZS0).

In step 19 the basic material-dependent routine, ITER, is called to separate the total incremental strains DE into their elastic, plastic, and creep portions. A number of different plasticity and creep parameters are used

and/or updated by ITER during each residual-force iteration. After the elastic strains have been determined, ITER also calculates end-of-increment stresses SIGMA1. Using these stresses, along with the start-of-increment values SIGMAO, the FORCE routine is called in step 20 to obtain the corresponding incremental nodal forces DP.

The residual (unbalanced) forces P are now computed, as the difference between the incremental applied loads PAR and the internal resisting forces DP. For degrees of freedom having prescribed displacements (determined by the KFD array code), no displacement errors exist and thus the corresponding components of P are set to zero. In step 21 the ERCOMP routine is called to compute a residual error norm ERR1 corresponding to the residual forces P. This norm is output onto the printer file.

If the residual norm ERR1 is less than the user specified value ERRMAX (and if a condition is satisfied which assures that enough iterations have been performed to allow recognition of the onset of plasticity and creep), then control is transferred outside the major iteration loop to step 23. If on the other hand the specified convergence criteria have not been satisfied another iteration is performed using the current residual forces P.

If the number of iterations performed equals the maximum user specified value NITER, then step 22, is performed to update the Jacobion (tangent stiffness) matrix to improve convergence. Five options are available for updating the Jacobian, as discussed in detail in the BOPACE theoretical manual. The option to be used is user defined by the variable SCODE. KKODE is an integer code

in common block GEN1, and specifies whether the elastic or the plastic stiffness contributions are to be formed and merged into a system matrix by routine MERGE. Routine MRTAPE adds the elastic and plastic merged matrices together where necessary to obtain a total merged matrix. DECOMP performs a Gauss decomposition of the merged Jacobian matrix. Control then returns to the iteration loop, and another set of iterations is performed as before. When variable NVP = number of Jacobian updates reaches the maximum user specified value MAXUP (usually 1), control is transferred to step 23 regardless of the degree of convergence.

In step 23 the end-of-increment results are computed and prepared for output. The vector PAR is set equal to any residuals P remaining from the present increment, so that these residuals may be combined with the next load increment in order to avoid cumulative loading errors. The cumulative internal forces PP and displacements QQ are updated by addition of incremental values DP and DQ, respectively. The incremental strains DE are converted to elastic values by subtracting out the plastic components DEP1 and creep components DEC1. The cumulative strains EET, EPT, ECT (elastic, plastic, creep respectively) are updated by adding the appropriate incremental values DE, DEP1, DEC1. The cumulative thermal strain SUMTS is updated by adding incremental values DETEMP. The cumulative effective plastic strains SUMPS and creep strains SUMCS are computed.

Step 24 then outputs the incremental results. Routine HEAD prints a heading and basic incremental and iterative variables. OUTPQ prints cumulative internal forces PP and displacements QQ. OUTE, OUTP and OUTC print elastic,

plastic and creep data, respectively. OUTS prints stress results, and OUTG prints a summary including effective strain values.

If a restart tape is to be written, BIGS is called in step 25 to add the data for the current increment to this tape.

This completes the load increment loop in the MAIN program, and control is transferred back to step 12. When all load increments have been run for the current problem, control is transferred to step 1 and a new problem is started.

3.0 SUBROUTINE DEFINITIONS

This section gives a list of BOPACE 2-D subroutines, given in the order in which they appear in the program deck. (The linear equation solver subroutines are documented separately, and in considerable detail in Section 6.) The documentation of BOPACE routines in this section lists in parentheses the arguments passed to each routine in the calling sequence, and also lists the variables which are passed by common blocks. The definition of each argument and common variable is given by comment statements in each subroutine.

ITER - (PCODE,MAXYC,NEL,IMAT,PTYPE,KTYPE,NEMOD,EMODX,EMODY,NPRAT,PRATX,PRATY, NITABX,NKTABX,NFTABX,NTABY,ITABX,KTABX,FTABX,TABY,ISTAB,KSTAB,FSTAB,EET,DE, T0,T1,YCODE0,YCODE1,YIELD0,YIELD1,TWORK0,TWORK1,TBASE0,TABASE1,EBASE0,EBASE1, SIGMA0,SIGMA1,ALPHA0,ALPHA1,DEP0,DEP1,CTYPE,NCREEP,CREEPX,CREEPY,CBASEX,NCTABX, NCTABY,CTABX,CTABY,CTAB,CTIME,CWORK0,CWORK1,CBASE0,CBASE1,DEC0,DEC1)

ITER is the major elastic-plastic-creep material-dependent routine. It takes the given incremental strains DE, and separates them into elastic, plastic DEP1, and creep DEC1, portions. It also computes end-of-increment stresses SIGMA1, and updates many of the incremental parameters. The ITER routine described here is based on the new "strain-space" algorithm (see addendum to the theoretical manual).

ITERO - (DEP1,ALPHA0,ALPHA1,EPSXX,EPSYY,EPSZZ,EPSXY,BETXX0,BETYY0,BETXY0, TS1,IS1,IE0,G0,G1,DC1,CEP,CR,LAMDA)

ITERO and ITER1 (given below) are called from the elastic-plastic-creep routine ITER, to compute an improved value of the plastic proportionality constant LAMDA, by the "linear intersection method" as described in the addendum to the BOPACE 2-D theoretical manual. These routines are in the new BOPACE version only. ITERO is called for plane stress problems, and ITER1 is called for plane strain problems, since the calculations involving the Z (normal) strain should be performed somewhat differently for the two cases in order to obtain the best convergence characteristics. The program would still operate quite well if the call statements to these two routines were eliminated from ITER, although the convergence would probably be solved somewhat.

<u>ITER1</u> - (DEP1,ALPHAO,ALPHA1,EPSXX,EPSYY,EPSZZ,EPSXY,BETXXO,BETYYO,BETXYO, TS1,IS1,IEO,GO,G1,DC1,LAMDA)

ITER1 is called for plane strain problems, and operates similarly to ITERO. ITER1 is somewhat simpler however because the Z-strain is known and does not have to be computed as in ITERO.

<u>CREEP</u> - (KSP,NEL,IMAT,CTYPE,NCREEP,CREEPX,CREEPY,CBASEX,NCTABX,NCTABY,CTABX, CTABY,CTAB,CTIME,TO,T1,SIGMA),SIGMA1,DECO,DEC1,CWORKO,CWORK1,CBASEO,CBASE1)

CREEP is a routine which was used in the initial BOPACE version to compute incremental creep strains DEC1. Its function is now performed by the new ITER routine, which computes both plastic and creep strains.

BIGS - (KODE,I1,I2,PCODE,NMAT,THICK,TEMPO,NTHERM,THERMX,THERMY,NEMOD,EMODX, EMODY,NPRAT,PRATX,PRATY,PTYPE,KTYPE,NITABX,NKTABX,NFTABX,NTABY,ITABX,KTABX, FTABX,TABY,ISTAB,KSTAB,FSTAB,COORDA,NOD,NEL,COORD,GCOS,IMAT,T,ELNO,NODE,NELE, NODI,NELI,KFD,PREF,YCODE1,YIELD1,TWORK1,TBASE1,EBASE1,T1,ZS1,SIGMA1,ALPHA1, DEPO,DEP1,EET,EPT,P1,PP,QO,PAR,CTYPE,NCREEP,CREEPX,CREEPY,CBASEX,NCTABX, NCTABY,CTABX,CTABY,CTAB,DECO,DEC1,CWORK1,CBASE1,ECT,SUMTS,SUMPS,SUMCS,NMAX1, NMAX2,NMAX3,NMAX4,NMAX5,NMAX6,NMAX7,NMAX8A,NMAX8B,NMAX8C,NMAX9,NMAX10,NMAX11, NMAX12,NMAX13)

BIGS performs three basic functions, depending on the value assigned to KODE. If KODE = 1 it sets up variables for a problem cold start, by calling several input routines and initializing values. If KODE = 2 it reads basic data from the restart tape and then searches through the load increment data to find and read data for the particular increment from the end of which a restart is to be made. If KODE = 3 it writes either basic data or data for a load increment onto the restart tape. Because of the large number of arguments passed and maximum 63 argument restriction on the UNIVAC 1108, the 1108 BOPACE version has many of the arguments put into common blocks.

GET - The GET routine was used in a low-core version of the BOPACE 1000-DOF program for the 1108, in order to keep core storage within the 64K word requirement. It groups arrays with major storage requirements into 12 groups. These groups are paired such that variables in one group are not needed by the program while the paired group is being used. The 6 pairs are stored on 6 files (A,B,C,D,E,F), and the GET routine is called at various points in the program to perform a "flip-flop" operation, i.e. bring one group into

core while writing its paired group out onto a storage file. This procedure is not especially efficient, although for large problems its relative cost is not great. A better procedure is the type of approach used in BOPACE 3-D in which data for each element is stored sequentially on a file. It is expected that the BOPACE 2-D program logic will eventually be revised to correspond to that of BOPACE 3-D. A more detailed description of the arrays involved and the file pairing techniques is given in Section 5.2.

<u>READRS</u> - (UIN1,UIN2,UOUT,INCR,UINRS,UOUTRS)

READRS is the first input routine called for each problem. It reads data file numbers and start-restart codes.

<u>READO</u> - (UI,UO,SCODE,MAXUP,MAXIT,MAXIE,MAXYC,MAXCUT,CUT,ERRMAX,AFACT)

READO reads the problem identification title, and various incremental and iterative constants. For constants not read (left blank by the user) default values are assigned.

<u>READ1</u> - (UI,UO,NMAX1,PCDOE,NMAT,THICK,TEMPO)

READ1 reads basic codes and constants.

<u>READTM</u> - (UI,UO,NMAX6,IMAT,PCODE,NTHERM,THERMX,THERMY,NEMOD,EMODX,EMODY, NPRAT,PRATX,PRATY)

READTM reads tables of thermal strain, elastic modulus, and Poisson's ratio, for each material as a function of temperature.

READTP - (UI,UO,NMAX7,NMAX8A,NMAX8B,NMAX8C,IMAT,PTYPE,KTYPE,NITABX,NKTABX,
NFTABX,NTABY,ITABX,KTABX,FTABX,TABY,ISTAB,KSTAB,FSTAB)

READTP reads the plastic hardening data, in the form of tables (ISTAB, KSTAB and FSTAB) of yield surface size, and shape and factor for yield surface translation, given as a function of temperature and hardening parameters. Although different abscissa (hardening parameter) values may be input for a material for each of its temperature values, READTP interpolates abscissas for all temperatures of the material to those of the first temperature given, in order to allow later use of an efficient table lookup procedure. Both the input and interpolated tables are output during the echo check of the input data.

READTC - (UI,UO,NMAX9,NMAX10,NMAX11,IMAT,CTYPE,NCREEP,CREEPX,CREEPY,CBASEX,
NCTABX,NCTABY,CTABX,CTABY,CTAB)

READTC logic is similar to that of READTP. It reads temperature value CTABY, stress levels CTABX, and creep factors CTAB.

READC - (UI,UO,NMAX12,COORDA)

READC reads angles of user defined special coordinate systems in degrees, and stores their values in radians in COORDA.

READM - (UI,UO,NMAT,NMAX2,NMAX3,NMAX4,NMAX5,NMAX12,THICK,NOD,NEL,COORDA,
COORD,GCOS,IMAT,T,ELNO,NODE,NELE,NODI,NELI)

READM reads the problem mesh data, including nodes and elements. Nodal and element input data in READM and other routines are identified by an associated node or element I.D. number. The internal program node and element numbers

are assigned sequentially in the order in which the node and element definition cards appear in the data deck. Correlation between I.D. and internal numbers is established by the vectors NODE,NELE,NODI, and NELI.

<u>READ2</u> - (UI,UO,NMAX4,NOD,NODI,NEL,ELNO,KFD)

READ2 reads the specified displacements and constraints, and places appropriate codes in the KFD vector. KFD(K) = +I for specified force, -I for specified displacement, and ±J for constrained degree of freedom. Here K denotes the degree of freedom, I is the node for freedom K is located, and J is the node of the independent freedom to which a corresponding freedom at node I is constrained to be equal.

<u>READ3</u> - (UI,UO,NMAX4,NMAX13,NOD,NODI,KFD,PREF)

READ3 reads the two load reference curves and places their values in the PREF array.

<u>READ4</u> - (UI,UO,NMAX14,MAXIT,NINCR,NITER,PFACT,CTIME)

READ4 reads incremental data (NINCR = number of load increments, NITER - maximum number of iterations per increment, PFACT = factors to be applied to load reference vectors, and CTIME = creep time increment). These data are read for all increments of a problem during a single call to READ4. They are checked and stored to avoid problem termination due to errors after several increments have been already solved.

READ5 - (UI,UO,INCR,NMAX5,NEL,NELE,NELI,TIDENT,TO,T1,ZSO,ZS1)

READ5 reads incremental temperature (thermal load) and Z-direction (normal) load data for each element. It is called at the beginning of each load increment. T1 is the new end-of-increment temperature vector, and ZS1 is the Z-load vector.

HEAD - (UO,NEL,INCR,F1,F2,CTIME,TIDENT,YCODEO,YCODE1,NITER,NI,MAXUP,NUP, ERRMAX,ERR)

HEAD writes a heading after each load increment is solved, giving values for basic incremental variables.

OUTE - (UO,NEL,NELE,DETEMP,SUMTS,DE,EET)

OUTE writes the incremental and cumulative values for thermal and elastic strains, at the end of each load increment.

OUTP - (UO,NEL,NELE,TWORKO,TWORK1,DEP1,EPT)

OUTP writes the incremental and cumulative values for plastic work density and plastic strains, at the end of each load increment.

OUTC - (UO,NEL,NELE,CWORKO,CWORK1,DEC1,ECT)

OUTC writes the incremental and cumulative values for creep work density and creep strains, at the end of each load increment where creep occurs.

OUTS - (UO,NEL,NELE,ALPHA1,SIGMA1)

OUTS writes end-of-increment values for stress centers ALPHA1 and stresses SIGMA1.

OUTG - (UO,NEL,NELE,YCODEO,YCODE1,TO,T1,YIELD1,DEP1,SUMPS,EPT,DEC1,SUMCS, ECT)

OUTG writes a summary of several incremental quantities, including effective plastic and creep strain values.

ROTK - (IT,ELNO,GCOS,K)

ROTK rotates the elemental stiffness matrix K from the global coordinates to user coordinates, using the direction cosines stored in GCOS.

ROTQ - (IT,ELNO,GCOS,KODE,Q)

ROTQ rotates the element displacements Q, either from nodal to global or global to nodal, depending on the code KODE.

STRAIN - (NEL,ELNO,COORD,GCOS,Q,ET)

STRAIN computes element strains ET in the element coordinate system (X-axis along nodes 1-2), from the nodal displacements Q.

FORCE - (NOD,NEL,T,ELNO,COORD,GCOS,SIGMAO,SIGMA1,P)

FORCE computes incremental forces P in the nodal coordinate system, from element incremental stresses (difference between end-of-increment SIGMA1 and start-of-increment SIGMAO stress values).

ERCOMP - (NEL,NN,KFD,ELNO,COORD,SIGMAO,SIGMA1,T,P,ERR,UO)

ERRCOMP computes a residual error norm. The norm is essentially a ratio of unbalanced forces P to total incremental forces, and is computed using the stresses SIGMAO and SIGMA1.

YVAL - (NP,AX,AY,X)

YVAL is a linear interpolation function routine. Given NP = number of points in curve, AX = vector of abscissas, AY = vector of ordinates, and X = desired abscissa value, YVAL is assigned the value of the interpolated ordinate corresponding to X. If X is outside the range of values in AX, the closest abscissa (first or last value of AX) is used instead. The values given in AX should be unique and monatonically increasing.

DYVAL - (NP,AX,AY,X0,X1)

DYVAL is an incremental interpolation function routine, and provides an incremental ordinate value equal to YVAL(NP,AX,AY,X1) minus YVAL(NP,AX,AY,X0).

ZVAL - (NTABX,NTABY,TABX,TABY,TABLE,X,Y)

ZVAL is a linear table interpolation function routine, which is a 2-dimensional version of YVAL. NTABX,NTABY are the number of points in the X,Y directions. TABX,TABY are the values in the X,Y directions. X,Y are the coordinates of the desired point, and ZVAL is assigned the corresponding interpolated TABLE (X,Y) value.

KEFORM - (T,E,NU,X21,X31,X32,Y21,Y31,Y32,K)

KEFORM forms the elastic stiffness matrix for the constant-strain-triangle element. It is called by routine GENER8.

<u>KPFORM</u> - (II,PCODE,IM,T,E,NU,X21,X31,X32,Y21,Y31,Y32,K)

```
COMMON/GENP0/AFACT,KTYPE
      /GENP1/SIGMA1
      /GENP2/ALPHA1
      /GENP3/TBASE0
      /GENP4/TBASE1
      /GENP5/EBASE1
      /GENP6/T1
      /GENP7/NITABX
      /GENP8/NKTABX
      /GENP9/NFTABX
      /GENP10/NTABY
      /GENP11/ITABX
      /GENP12/KTABX
      /GENP13/FTABX
      /GENP14/TABY
      /GENP15/ISTAB
      /GENP16/KSTAB
      /GENP17/FSTAB
      /GENP18/DEP1
```

KPFORM forms the plastic contribution to the stiffness matrix for the constant-strain triangle element. It is called by routine GENER8. The common variables are used in the elastic-plastic material theory. The plastic theory follows Section 2 of the theoretical manual.

GENER8 - (IT,NUM,NODES,K,NK)

COMMON/GEN1/PCODE,KKODE

/GEN2/YCODE1

/GEN3/IMAT

/GEN4/T

/GEN5/ELNO

/GEN6/COORD

/GEN7/GCOS

/GEN8/NEMOD,EMODX,EMODY,NPRAT,PRATX,PRATY

/GENP6/T1

GENER8 is the basic elemental stiffness generator, which calls either KEFORM or KPFORM to form the elastic or plastic stiffness matrix. GENER8 is called by routine GENR8 in the linear equation solver package. The stiffness for plane stress or plane strain is formed depending on whether the value of PCODE is 0 or 1, respectively. For plane strain the value of E = elastic modulus and NU = Poisson's ratio are adjusted so as to allow the same elastic stiffness generation procedure for both plane stress and plane strain.

4.0 INPUT/OUTPUT AND COMMON STORAGE

4.1 INPUT/OUTPUT FILES

The following is a definition of I/O files used in the BOPACE 2-D program, given by unit number (integer constant or variable).

5 - Input card file.

6 - Output printer file.

UIN1 - UIN1 is the user-defined unit number for the type I input data (see user manual, Section 2.0).

UIN2 - UIN2 is the user-defined unit number for the type II input data.

UOUT - UOUT is the user-defined unit number for the major output data file.

UINRS - UINRS is the user-defined unit number for the input restart data file.

UOUTRS - UOUTRS is the user-defined unit number for the output restart data file.

*UNITE1 = 11 - File for storing the merged elastic stiffness matrix.

*UNITE2 = 12 - File for storing the decomposed elastic stiffness matrix.

*Note: UNITE1,UNITE2,UNITP1,UNITP2 may also be used as temporary scratch files when adding elastic and plastic matrix contributions to form the total stiffness.

*UNITP1 = 13 - File for storing the merged total Jacobian matrix. It is used only when the input variable SCODE is equal to 3, 4 or 5.

*UNITP2 = 14 - File for storing the decomposed total Jacobian matrix. It is used only when the input variable SCODE is equal to 3, 4 or 5.

UNITS1 = 15 - Scratch file used for temporary storage by the Gauss merge and decomposition routines.

UNITS2 = 16 - Same as UNITS1.

20-26 - These units provide storage for the "flip-flop" arrays handled by routine GET (see Section 5.2). Six files are used for six paired array groups, plus one file for temporary storage.

4.2 COMMON BLOCKS

The only common blocks used in the current BOPACE 2-D program version occur in the MAIN program, in the linear equation solver routines (see Section 6), and in routines KPFORM and GENER8. (In the UNIVAC 1108 version some of the variables in routine BIGS have also been placed in common to reduce the length of the argument list).

The common block JLB in the MAIN program provides for a storage area needed in solving the linear equations, and for transfer of variables between the MAIN program and the equation solver routines.

*Note: UNITE1,UNITE2,UNITP1,UNITP2 may also be used as temporary scratch files when adding elastic and plastic matrix contributions to form the total stiffness.

Common blocks GEN1,GEN2,GEN3,GEN4,GEN5,GEN6,GEN7,GEN8 contain basic codes, element and nodal data, and elastic material properties, which are used in the GENER8 routines.

Common blocks GENP0,GENP1,GENP2,GENP3,GENP4,GENP5,GENP6,GENP7,GENP8,GENP9, GENP10,GENP11,GENP12,GENP13,GENP14,GENP15,GENP16,GENP17,GENP18, are plasticity variables and data tables, which are used to form the plastic stiffness matrices in routine KPFORM. The temperatures stored in GENP6 are also used in routine GENER8.

5.0 CORE REDUCTIONS

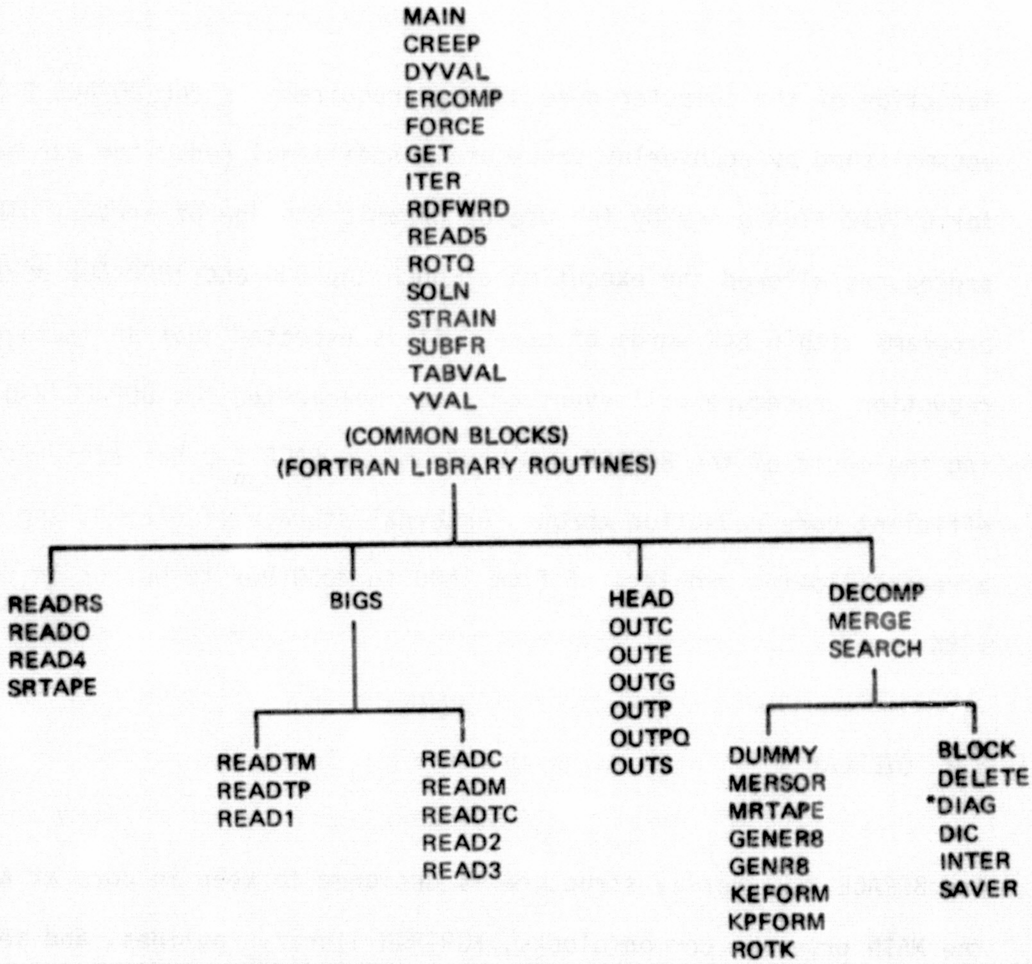
Reduction of the computer core storage requirements for BOPACE 2-D is accomplished by an overlay procedure. Additional reduction was achieved for UNIVAC 1108 usage by the use of dynamic storage of arrays. These procedures allowed the execution of both the 300 and 1000 DOF BOPACE 2-D programs within 64K words of core. It is expected that an improved core reduction procedure will eventually be implemented for BOPACE 2-D, following the logic of the BOPACE 3-D program. BOPACE 3-D has achieved a very efficient core reduction through external storage of element and nodal arrays, allowing problems of from 1500 to 3000 DOF to be run in 64K to 128K core.

5.1 OVERLAY

The BOPACE 2-D overlay structure is designed to keep in core at all times the MAIN program, common blocks, FORTRAN library routines, and several subroutines which are called within the MAIN program inner loop. For those routines which are overlayed, two overlay levels (A and B) are defined. The overlay is diagrammed in Figure 5.1-0.

5.2 DYNAMIC STORAGE OF ARRAYS

This procedure involves grouping of the major program arrays into 12 groups, and pairing them onto 6 files (A,B,C,D,E,F). The groups are transferred into and out of core by routine GET (see Section 3.0). For example, while group A1 is in core, group A2 is stored on file A, etc. The following



* INCLUDES DROW AND DOTHER

Figure 5.1-0. BOPACE 2-D Overlay Diagram

defines the array groupings, variable dimensions, and resulting group storage lengths.

A1(7000)-P0(1000),P1(1000),DP(1000),DQ(1000),PP(1000),QQ(1000),PAR(1000)

A2(7200)-YCODE1(800),TWORK1(800),T0(800),T1(800),TBASE0(800),TBASE1(800),
ISTAB(900),KSTAB(600),FSTAB(900)

B1(7200)-SUMTS(800),SUMPS(800),SUMCS(800),EPT(2400),ECT(2400)

B2(7200)-IMAT(800),ALPHA1(2400),SIGMA1(3200),EBASE1(800)

C1(6600)-STOR(5000),ZS0(800),ZS1(800)

C2(6700)-DETEMP(800),YCODE0(800),EBASE0(800),YIELD1(800),SIGMA0(3200),CTAB(300)

D1(5800)-GCOS(2000),COORD(1000),PREF(2000),NELE(800)

D2(5600)-YIELD0(800),ALPHA0(2400),DEC0(2400)

E1(6600)-T(800),ELNO(2400),DEC1(2400),KFD(1000)

E2(6400)-TWORK0(800),DEP0(2400),EET(3200)

F1(5700)-CWORK0(800),CWORK1(800),CBASE0(800),CBASE1(800),P(1000),Q(1000),
NODE(500)

F2(5600)-DEP1(2400),DE(3200)

6.0 LINEAR EQUATION SOLVER

6.1 INTRODUCTION

These routines are written in FORTRAN IV for use on IBM 360 and 370, and UNIVAC 1108 computers.

6.1.1 PURPOSE

These routines are used to: 1) generate elemental matrices and merge them into the gross matrix; 2) decompose the gross matrix; 3) forward and back substitute to find the unknowns (see Section 5 of the theory document); and 4) two special routines, one to merge two gross matrices, and one to read and/or write a checkpoint tape containing the gross matrix and the decomposed matrix. The routines are separated into the above logical sections in order to give the user complete flexibility in their use based on the type of problem to be solved.

The linear equation solver routines have been written as an independent package. As such, they can be used in any program without recoding. The user must supply routines to generate the elemental matrices (details given in Section 6.2), which enhances their independence and use.

6.1.2 NOMENCLATURE

NF - Number of freedoms per node

NN - Number of nodes in problem

F - Input - vector of combined known forces and displacements
Output - vector of known and calculated (reactions) forces

D - Vector of known and calculated displacements

6.1.2 (continued)

MBW - Maximum bandwidth

LDT - Load definition vector

6.1.3 METHOD

6.1.3.1 MATRIX PARTITIONING

Figure 6.1-1 shows a two bar, three node structure; each bar is capable of carrying axial loads.

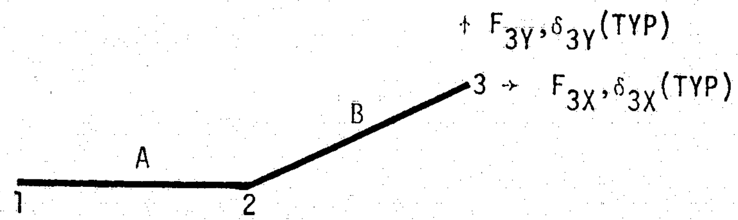


FIGURE 6.1-1: TWO BAR STRUCTURE

Bar A, using nodes 1 and 2, has the elemental matrix given in Equation (6.1-1). Note that bar A is parallel to the X axis and does not have Y freedoms.

$$\begin{Bmatrix} F_{1X} \\ F_{2X} \end{Bmatrix} = \begin{bmatrix} K & K \\ K & K \end{bmatrix} \begin{Bmatrix} \delta_{1X} \\ \delta_{2X} \end{Bmatrix} \qquad (6.1\text{-}1)$$

where K represents the stiffness terms (for this discussion their value is immaterial).

Bar B, using nodes 2 and 3, has the elemental matrix given in Equation (6.1-2). Note that bar B has both X and Y freedoms.

6.1.3.1 (continued)

$$\begin{Bmatrix} F_{2X} \\ F_{2Y} \\ F_{3X} \\ F_{3Y} \end{Bmatrix} = \begin{bmatrix} K & K & K & K \\ K & K & K & K \\ K & K & K & K \\ K & K & K & K \end{bmatrix} \cdot \begin{Bmatrix} \delta_{2X} \\ \delta_{2Y} \\ \delta_{3X} \\ \delta_{3Y} \end{Bmatrix} \qquad (6.1\text{-}2)$$

Merging these two elemental matrices yields the gross matrix

$$\begin{Bmatrix} F_{1X} \\ F_{2X} \\ F_{2Y} \\ F_{3X} \\ F_{3Y} \end{Bmatrix} = \begin{bmatrix} K & K & 0 & 0 & 0 \\ K & K & K & K & K \\ 0 & K & K & K & K \\ 0 & K & K & K & K \\ 0 & K & K & K & K \end{bmatrix} \begin{Bmatrix} \delta_{1X} \\ \delta_{2X} \\ \delta_{2Y} \\ \delta_{3X} \\ \delta_{3Y} \end{Bmatrix} \qquad (6.1\text{-}3)$$

The program requires that each node has the same number of freedoms in the matrices. Therefore Equation (6.1-1) must be rewritten

$$\begin{Bmatrix} F_{1X} \\ F_{1Y} \\ F_{2X} \\ F_{2Y} \end{Bmatrix} = \begin{bmatrix} K & 0 & K & 0 \\ 0 & 0 & 0 & 0 \\ K & 0 & K & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \delta_{1X} \\ \delta_{1Y} \\ \delta_{2X} \\ \delta_{2Y} \end{Bmatrix} \qquad (6.1\text{-}4)$$

and the gross matrix becomes

$$\begin{Bmatrix} F_{1X} \\ F_{1Y} \\ F_{2X} \\ F_{2Y} \\ F_{3X} \\ F_{3Y} \end{Bmatrix} = \begin{bmatrix} K & 0 & K & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ K & 0 & K & K & K & K \\ 0 & 0 & K & K & K & K \\ 0 & 0 & K & K & K & K \\ 0 & 0 & K & K & K & K \end{bmatrix} \begin{Bmatrix} \delta_{1X} \\ \delta_{1Y} \\ \delta_{2X} \\ \delta_{2Y} \\ \delta_{3X} \\ \delta_{3Y} \end{Bmatrix} \qquad (6.1\text{-}5)$$

The gross matrix of Equation (6.1-5) can be partitioned

$$\left\{\begin{matrix} \begin{Bmatrix} F_{1X} \\ F_{1Y} \end{Bmatrix} \\ \begin{Bmatrix} F_{2X} \\ F_{2Y} \end{Bmatrix} \\ \begin{Bmatrix} F_{3X} \\ F_{3Y} \end{Bmatrix} \end{matrix}\right\} = \begin{bmatrix} \begin{bmatrix} K & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} K & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \begin{bmatrix} K & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} K & K \\ K & K \end{bmatrix} & \begin{bmatrix} K & K \\ K & K \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} K & K \\ K & K \end{bmatrix} & \begin{bmatrix} K & K \\ K & K \end{bmatrix} \end{bmatrix} \left\{\begin{matrix} \begin{Bmatrix} \delta_{1X} \\ \delta_{1Y} \end{Bmatrix} \\ \begin{Bmatrix} \delta_{2X} \\ \delta_{2Y} \end{Bmatrix} \\ \begin{Bmatrix} \delta_{3X} \\ \delta_{3Y} \end{Bmatrix} \end{matrix}\right\} \qquad (6.1\text{-}6)$$

The reason for the programs requirement that each node have the same number of freedoms becomes apparent when examining the partitions of Equation (6.1-6). Note that each partition has the same matrix order (in this case a 2X2), also that the freedoms within the partition are ordered identically. Further, the location of each partition is known by identifying the node number associated with its rows and the node number associated with its columns.

The program takes advantage of these conditions for several reasons. First, by generating the full elemental matrices (as shown in Equations (6.1-2) and (6.1-4)), and doing the partitioning on them, makes merging to form the gross matrix a simple addition of partitions having identical row/column node numbers. Second, the special equation generation process described in Section 5 of the theory manual becomes a matter of how the elemental matrices are partitioned. (And the row/column node numbers assigned.) For example, assume that freedom Y, node 3 of bar B (see Figure 6.1-1) is

constrained to be equal to freedom Y of node 4 (not shown). Then rewritten Equation (6.1-2) (with partitioning) yields

$$\begin{Bmatrix} \begin{Bmatrix} F_{2X} \\ F_{2Y} \end{Bmatrix} \\ \begin{Bmatrix} F_{3X} \\ F_{3Y} \end{Bmatrix} \\ \begin{Bmatrix} F_{4X} \\ F_{4Y} \end{Bmatrix} \end{Bmatrix} = \begin{bmatrix} \begin{bmatrix} K & K \\ K & K \end{bmatrix} & \begin{bmatrix} K & 0 \\ K & 0 \end{bmatrix} & \begin{bmatrix} 0 & K \\ 0 & K \end{bmatrix} \\ \begin{bmatrix} K & K \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} K & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & K \\ 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ K & K \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ K & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & K \end{bmatrix} \end{bmatrix} \begin{Bmatrix} \begin{Bmatrix} \delta_{2X} \\ \delta_{2Y} \end{Bmatrix} \\ \begin{Bmatrix} \delta_{3X} \\ \delta_{3Y} \end{Bmatrix} \\ \begin{Bmatrix} \delta_{4X} \\ \delta_{4Y} \end{Bmatrix} \end{Bmatrix} \tag{6.1-7}$$

Third, the program needs to retain only those partitions which have at least one non-zero term. Further, due to symmetry of the stiffness matrix, it retains only the upper triangular partitions (those whose row node numbers are less than or equal to the column node numbers).

Fourth, and finally, the decomposition, and forward/backward substitution process becomes a matter of operating on the non-zero partitions. This in turn simplified the inherent bookkeeping necessary during these steps. It cuts down the core requirements, particularly during the decomposition for the matrix, and also for the bookkeeping arrays.

The row/column node numbers are packed into one FORTRAN integer. This partition ID number has the form

$$ID = f * I + J \tag{6.1-8}$$

where ID = the ID number

f = a factor (currently f = 1001)

I = row node number

6.1.3.1 (continued)

The use of this ID number allows the merging routine to: 1) search only a one dimensional vector of ID numbers in order to merge the partitions, and 2) to order the partitions on the merged matrix tape (disk, drum, etc.) by low ID to high ID numbers. This means that the partitions are ordered on the tape low row node numbers to high row node numbers. And within each group of row node numbered partitions, the partitions are ordered low column node number to high column node numbers. The value of a sorted merged matrix tape is used to great advantage in calculating the maximum bandwidth (Subroutine DUMMY) and the incore decomposition (Subroutine DIC).

6.1.3.2 NODE NUMBERING

These routines require that the node numbers are FORTRAN integers in the range 1 to NN, where NN is the number of nodes in the problem. This requirement allows the program to use the node number as a FORTRAN index in locating necessary data in the vectors (F, D, and LDT).

For example, assume that it is necessary to obtain the value of the J^{th} freedom of the I^{th} node of vector V. Then the following FORTRAN code can be used

```
      DIMENSION V(NF,NN), W(1)
      EQUIVALENCE (V(1,1), W(1))
    1 U = V(J,I)
    2 U = W(NF*(I-1) + J)
```

These routines use the second form (statement 2).

6.1.3.3 LOAD DEFINITION VECTOR

The load definition vector contains one value for each freedom (NF*NN values) in the problem. Its purpose is to describe the constraints of the freedoms. It is a FORTRAN integer vector, setup with node numbering and freedom storage as described in Section 6.1.3.2. The values assigned for each freedom must be assigned according to the following rules. (Let K equal the assigned value.)

1) +K, a positive value describes a freedom with a known force and the displacement is to be calculated
2) -K, a negative value describes a freedom with a known displacement and the reaction force is to be calculated.
3) The value K is a node number, and in general is the node number of the freedom, i.e., LDT(J,I) = $\pm$I where J = freedom and I = node number. For the special equation generation case (Section 5 of the theory document) the value of K at node I, freedom J, becomes the connecting node to tie the freedoms together. To clarify, consider the structure shown in Figure (6.1-2), consisting of three bars, capable of carrying axial loads only, and five nodes.

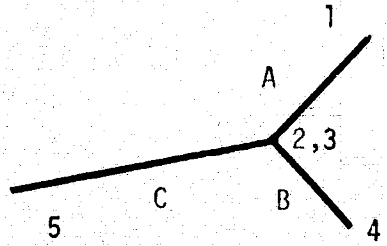


FIGURE 6.1-2: THREE BAR STRUCTURE

6.1.3.3 (continued)

Bar A on nodes 1 and 2, bar B on nodes 3 and 4, and bar C on nodes 3 and 5. Further assume that the X freedom at nodes 2 and 3 is constrained to displace equally, and the Y freedoms are to be allowed to displace unequally. The load definition vector (LDT) for this problem is given in Table 6.1-1.

NODE	FREEDOM	LDT	NODE	FREEDOM	LDT
1	X	1	4	X	-4
	Y	1		Y	-4
2	X	2	5	X	-4
	Y	2		Y	5
3	X	2			
	Y	3			

TABLE 6.1-1

One essential point - the cross connection is on an identical freedom basis, a freedom X to a freedom X, etc., but not a freedom Y to a freedom X.

Two powerful uses of the load definition vector are available. First, in the case of known displacements, the nodes do not need to be attached, (see nodes 4 and 5 above). Second, the known forces at the connected nodes are additive (an input force at 2X and an input force at 3X are added in the above example).

6.1.4 LIMITATIONS

The maximum core usage occurs during the incore decomposition. The available core is defined by the user in a named common block. One value in this common block is the length of the common block. (Details of this common

6.1.4 (continued)

block are given in Section 6.2.1).

Let MBW = maximum bandwidth

NF = number of freedoms per node

N = ((MBW)*(MBW+1)) / (2)

L = (MBW) + (MBW)*(NF^2) + (4)

C = required core (words)

Then

$$C = (MBW) + (N) + (2)*(NF^2)*(N) + (6)*(NF^2) + (L)$$

Currently the only available decomposition routine requires in-core storage having the above storage requirement. An out-of-core or large decomposition routine was planned but not implemented (to date).

One other decomposition limitation is the maximum size of L. This maximum is hard coded in the program (see "MAXLT" in subroutine DECOMP) and is set to 420 words.

6.2 USER INTERFACE

6.2.1 USAGE

While there are some twenty-four routines in the Linear Equation Solver package, the full capability is available via five "main" entry routines.

1) MERGE - to generate and merge the elemental matrices to obtain the gross matrix
2) DECOMP - to decompose the gross matrix
3) SOLN - to calculate the unknown forces and displacements
4) MRTAPE - to merge two gross matrices (from MERGE) using the equation

   $$[K_M] = A*[K_1] + B*[K_2]$$

   where

   $[K_M]$ is the new merged matrix

   $[K_1]$ is the first input gross matrix

   $[K_2]$ is the second input gross matrix

   A is an user supplied scalar

   B is an user supplied scalar
5) SRTAPE - to write or read a checkpoint tape

Details of the calling sequences is given in Section 6.2.2.

The gross matrix (or matrices) and the decomposed matrix are stored on an I/O unit (tape, disk, drum, etc.). The user is responsible for assignment of the FORTRAN logical units.

6.2.1 (continued)

The core storage used is a named common block. This allows the user complete control of core storage requirements for his program. The format of the common block is

```
COMMON/JLB/KT, LT, MBW, LA, A(LA)
```

where KT and LT are the FORTRAN logical unit numbers of two I/O units to be used as scratch

MBW is the maximum bandwidth to be expected during decomposition. This value is calculated by the Generate/Merge section or the MRTAPE routine and posted here.

LA is the length in words (the dimension) of array A.

A is the scratch storage array of length LA.

The node numbers are required to be in the range 1 to NN as described in Section 6.1.3.2.

Three arrays are used, they must be in user storage (not in common block /JLB/). The first array is the Load Definition Vector described in Section 6.1.3.3. The two remaining vectors are FORTRAN single precision real (floating point) vectors setup with node numbering and freedom storage as described in Section 6.1.3.2. These two arrays are the Force vector (F) and the Displacement vector (D). They are used only by the Forward/Backward Substitution section (Subroutine SOLN).

The Force vector (F) has dual usage; on input to SOLN, it must contain the combined known forces and displacements. On return from SOLN, it will contain only forces (known forces and calculated reactions).

6.2.1 (continued)

The Displacement vector (D) is not used for input to SOLN; on return, it will contain the displacements (known and calculated).

In order to give the user complete freedom of type of structural elements to be used, the Linear Equation Solver routines require the user to supply his own elemental matrix generation routines. Interface between the Generation/Merge section and the user routines is via the user written subroutine GENER8. Details are given in Section 6.2.3.

6.2.2 CALLING SEQUENCE

The calling sequence to the entry routines follows. Unless otherwise noted, all calling sequence arguments are fullword integer values or arrays.

CALL MERGE (IT,NN,NE,NF,LEN,LDT)

where

IT = FORTRAN logical unit number of the unit for the output gross matrix.

NN = number of nodes

NE = number of structural elements

NF = number of freedoms per node

LEN = the maximum number of nodes that can be generated for any one structural element

LDT = the Load Definition Vector

6.2.2 (continued)

CALL DECOMP (IT,JT,NF,LDT)

where

IT = FORTRAN logical unit number of the input gross matrix

JT = FORTRAN logical unit number of the unit for the output decomposed matrix

NF = number of freedoms per node

LDT = the Load Definition Vector

CALL SOLN (JT,NF,NN,LDT,F,D)

where

JT = FORTRAN logical unit number of the input decomposed matrix

NF = number of freedoms per node

NN = number of nodes

LDT = the Load Definition Vector

F = single precision real (floating point) array of the input combined known force and displacement vector, and the output forces

D = single precision real (floating point) array of the output displacement vector.

CALL MRTAPE (IT,JT,KT,A,B,NF)

where

IT = FORTRAN logical unit number of the first input gross matrix

JT = FORTRAN logical unit number of the second input gross matrix

A = single precision real (floating point) value, scale factor for the first input matrix on unit IT

6.2.2 (continued)

B = single precision real (floating point) value, scale factor for the second input matrix on unit JT

NF = number of freedoms per node

CALL SRTAPE (IT,JT,NF,IP,IS)

where

IT = FORTRAN logical unit number of the input gross matrix or the input decomposed matrix

JT = FORTRAN logical unit number of the unit for the checkpoint tape

NF = number of freedoms per node

IP = 1 process a gross matrix

= 2 process a decomposed matrix

IS = 1 write the checkpoint tape

= 2 read the checkpoint tape

SRTAPE must be called for each matrix to be read or written. It does not position the checkpoint tape other than the inherent positioning caused by the read/write operations. The user is required to position the tape as necessary. The no positioning concept allows the user to use the tape for checkpoint of his own data.

6.2.3 USER ROUTINES

The user is required to supply his own elemental matrix generator routines. Interface is via a user supplied routine with the specific name GENER8, which has the specific calling sequence.

6.2.3 (continued)

CALL GENER8 (I,N,NODES,S,NS)

where

- I = fullword integer value containing the element number whose elemental matrix is to be supplied (I is in the range 1 to NE where NE is the number of elements)
- N = fullword integer value to be returned containing the number of nodes of the element
- NODES = fullword integer array to be returned containing the N node numbers of the element. (Use the first N terms of this array.)
- S = double precision real (floating point) array to be returned containing the elemental matrix.
- NS = dimensions of array S.

Array S is dimensioned

```
DOUBLE PRECISION S(NS,NS)
```

The elemental matrix must be a full symmetric matrix (do not return a symmetric half). The nodes and freedoms are to be ordered such that the elemental matrix partitions can be formed directly from the elemental matrix, with no sorting or inserting of rows and columns to meet the modal partition order.

The node ordering of the matrix in array S must match the node ordering in array NODES.

Subroutine GENER8 is called once for each element in the problem. Its content, logic, and programming is left to the user.

6.3 PROGRAMMING

6.3.1 TAPE FORMATS

Two tape formats are used. The first format is used for the matrix partitions including all scratch tapes and the final gross matrix tape. The second format is for the decomposed matrix tape.

Figure 6.3-1 is the tape format for matrix partitions

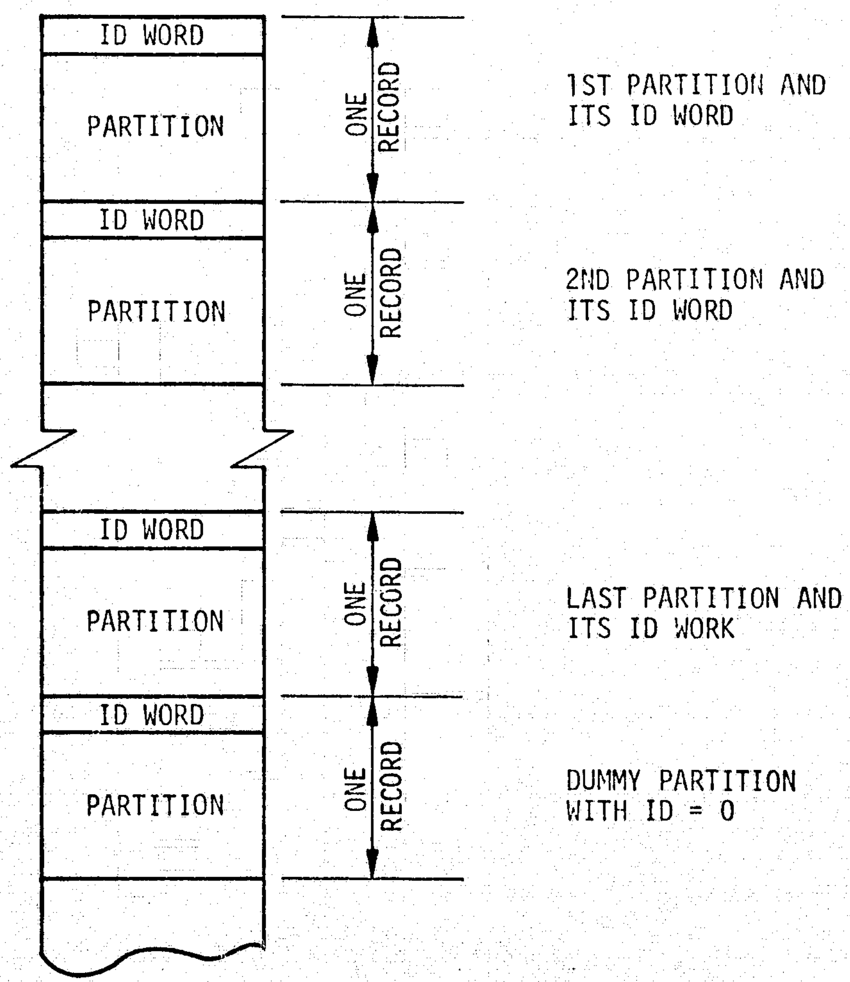


FIGURE 6.3-1: PARTITIONED MATRIX TAPE FORMAT

6.3.1 (continued)

Let NF be the number of freedoms per node. Then each record consists of the partition's ID (row/column packed node numbers) FORTRAN integer value (one full word) followed by the NF^2 double precision values of the elements of the partition. The elements of the partitions are written by rows, the NF elements of row one, followed by the NF elements of row two, etc. The physically last record on the tape must be a dummy partition with an ID number of zero.

Figure 6.3-2 shows the general tape format, and the format of one general data record. Each record on the tape, including the header and trailer records, has a FORTRAN fullword integer as its first word. This integer (NNW) is the number of words in the record.

The header and trailer records are each ten FORTRAN integer fullwords with the format shown in Figure 6.3-3.

Each data record contains one or more nodal records, each nodal record having the format shown in Figure 6.3-4.

Each nodal record contains all the data necessary from the decomposition step in order to perform the forward/backward substitution step. Each contains the data for one decomposition row. The partitions are single precision real (floating point) values, all other values are FORTRAN fullword integers. The nodes array contains the column node numbers of the partitions, with the first node also being the node number of the decomposition row.

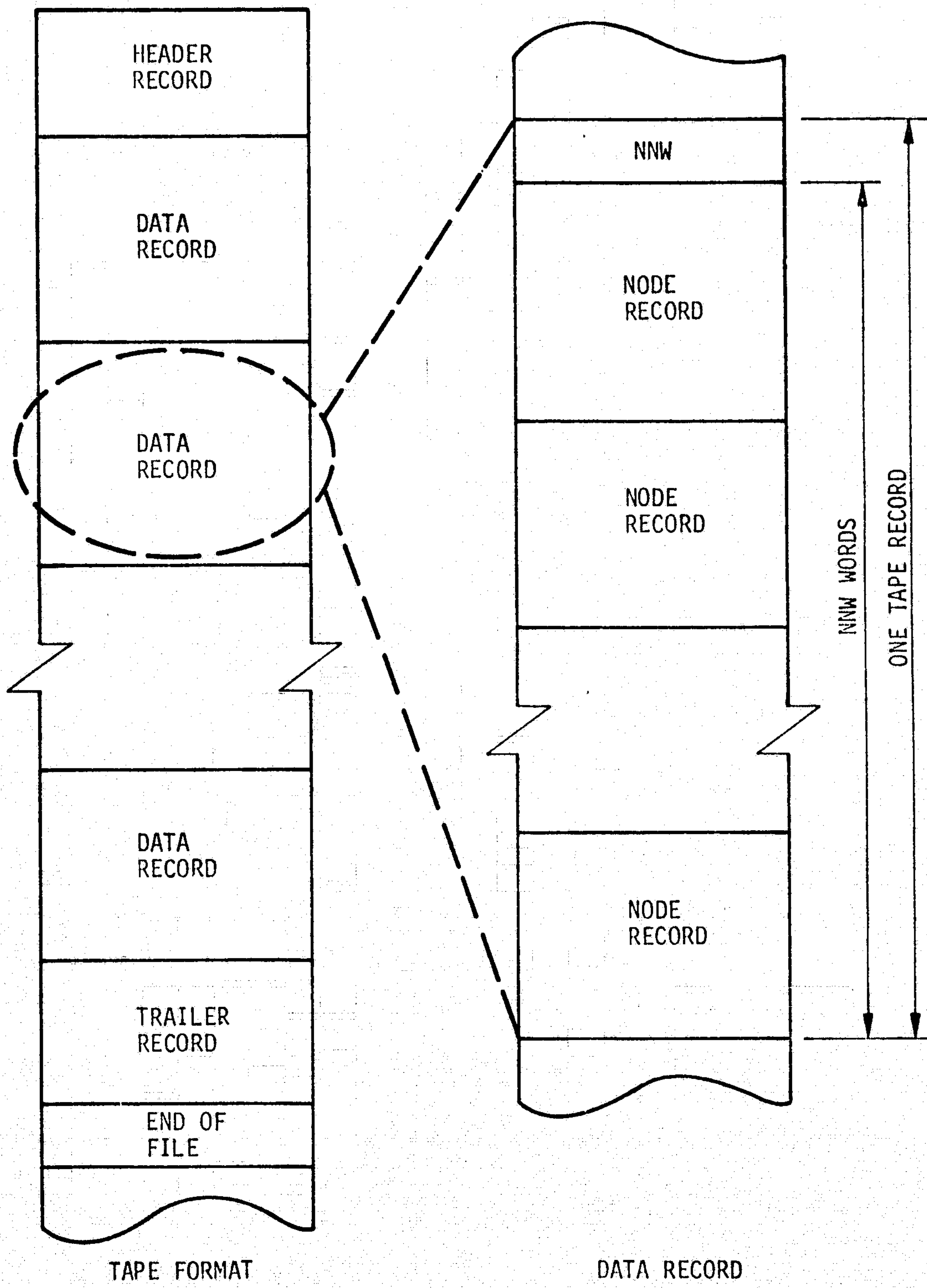
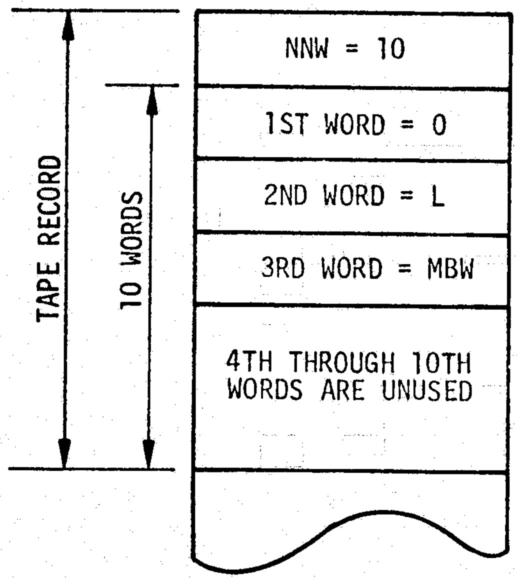


FIGURE 6.3-2: DECOMPOSITION TAPE FORMAT

6.3.1 (continued)

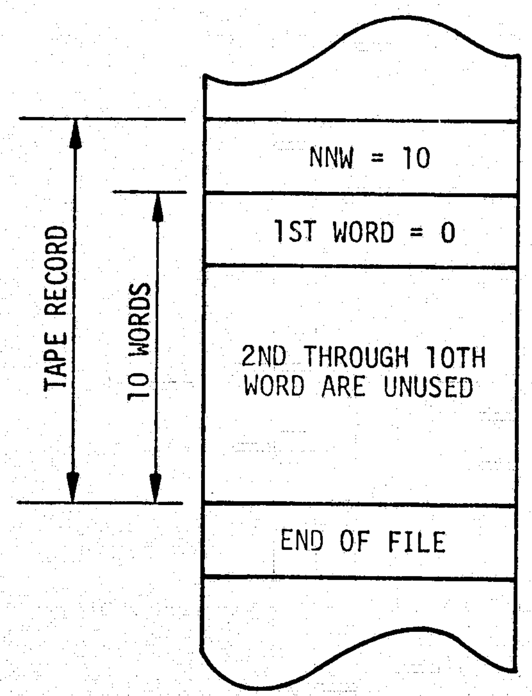
As many nodal records are placed in each data record as possible without exceeding the maximum length of a data record. Each nodal record is completely contained in its data record.



L = MAXIMUM LENGTH OF ANY ONE DATA RECORD (SEE MAXLT IN SUBROUTINE DECOMP)

MBW = MAXIMUM BANDWIDTH

HEADER RECORD



TRAILER RECORD

FIGURE 6.3-3: HEADER AND TRAILER RECORDS FORMAT

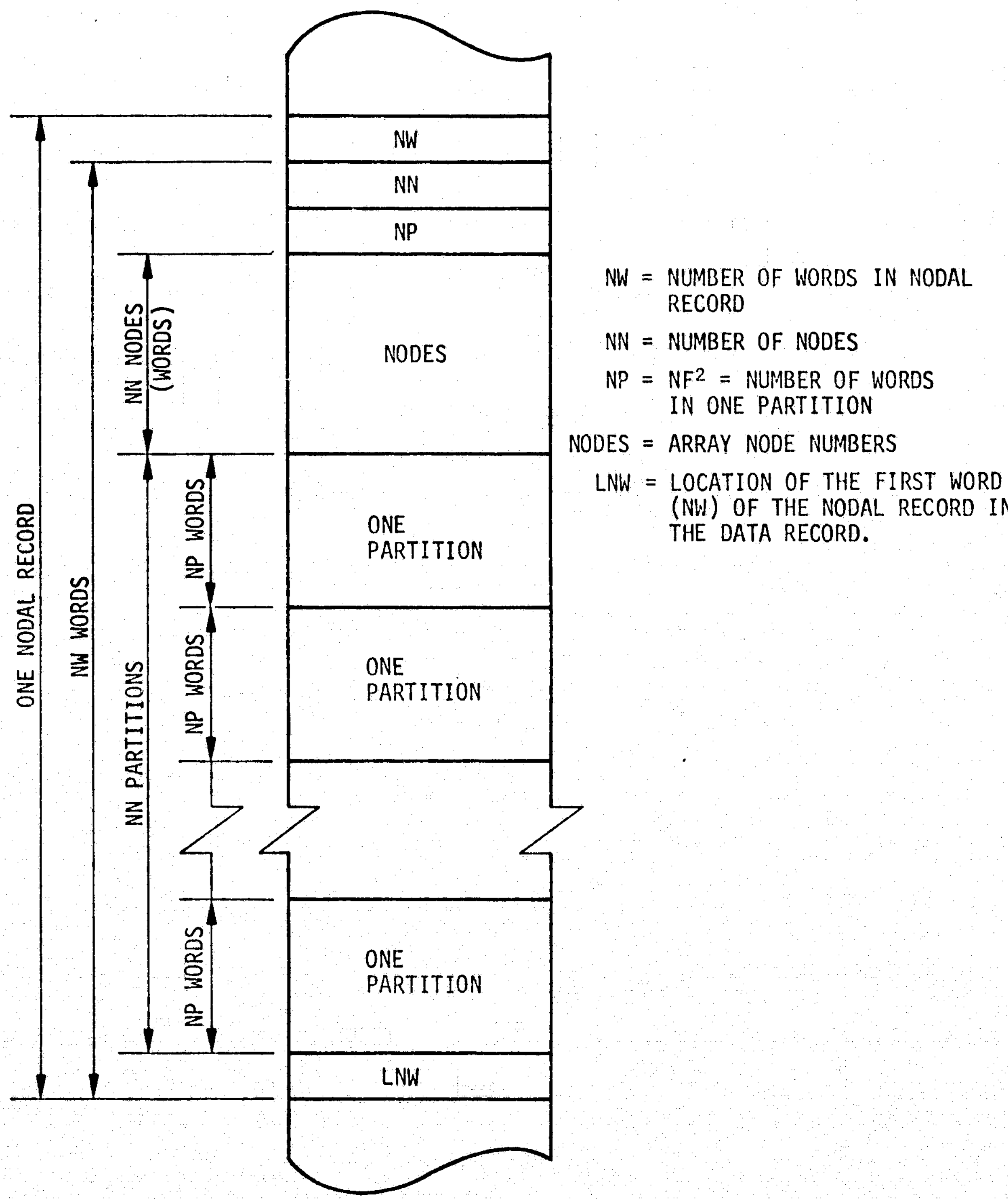


FIGURE 6.3-4: NODAL RECORD

6.3.2 GENERATE AND MERGE ROUTINES

The general flow of the GENERATE and MERGE routines are shown in Figure 6.3.5.

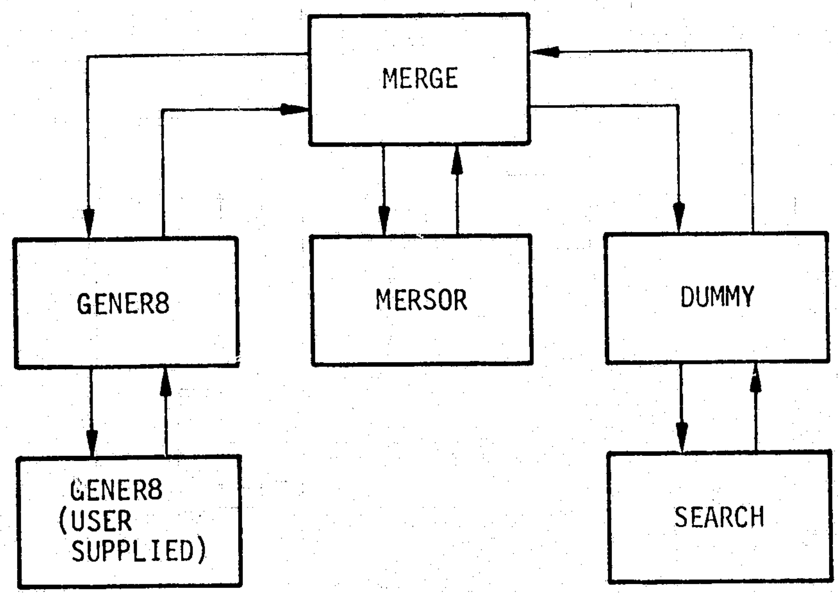


FIGURE 6.3-5: GENERAL FLOW GENERATE/MERGE ROUTINES

MERGE is the "main" entry subroutine called by the user. Its main function is to compute storage assignments for the arrays needed in the other routines. These storage assignments are in the common block /JLB/ and are passed via the calling sequence arguments.

GENR8 controls the generation of the elemental matrices. It partitions the elemental matrices, writting random ordered, unmerged partitions on a tape for later merging.

6.3.2 (continued)

MERSOR merges the partitions generated by GENR8 and writes the sorted merged gross matrix on the gross matrix tape.

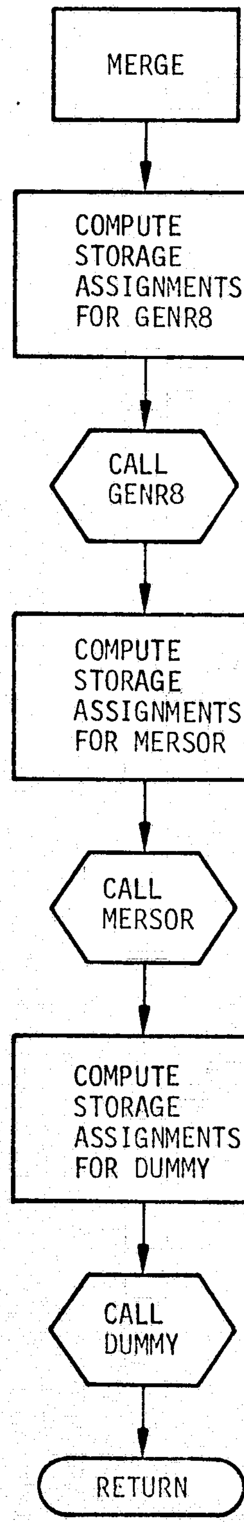
DUMMY computes the maximum bandwidth using the merged gross matrix tape from MERSOR.

SEARCH is a working routine used to locate a node number in an array of node numbers; it is also used by the decomposition step and the special routine MRTAPE.

6.3.2 (continued)

Subroutine MERGE

Calling sequence - see Section 6.2.2.



6.3.2 (continued)

Subroutine GENR8

Calling Sequence

```
CALL GENR8(ITAPE,NE,NF,NSN,LDT,S,B,C,LTT,NODES,LDD)
```

when

ITAPE = FORTRAN logical unit number of the unit for the generated partitions

NE = number of structural elements

NF = number of freedoms per node

NSN = dimension of array S

= LEN*NF where LEN = maximum number of nodes in any one element (see calling sequence for MERGE, Section 6.2.2)

LDT = Load Definition vector

S = generated matrix (see calling sequence to GENER8, Section 6.2.3)

B = array for one partition

C = array for one partition

LTT = array for Load Definition vector for one element

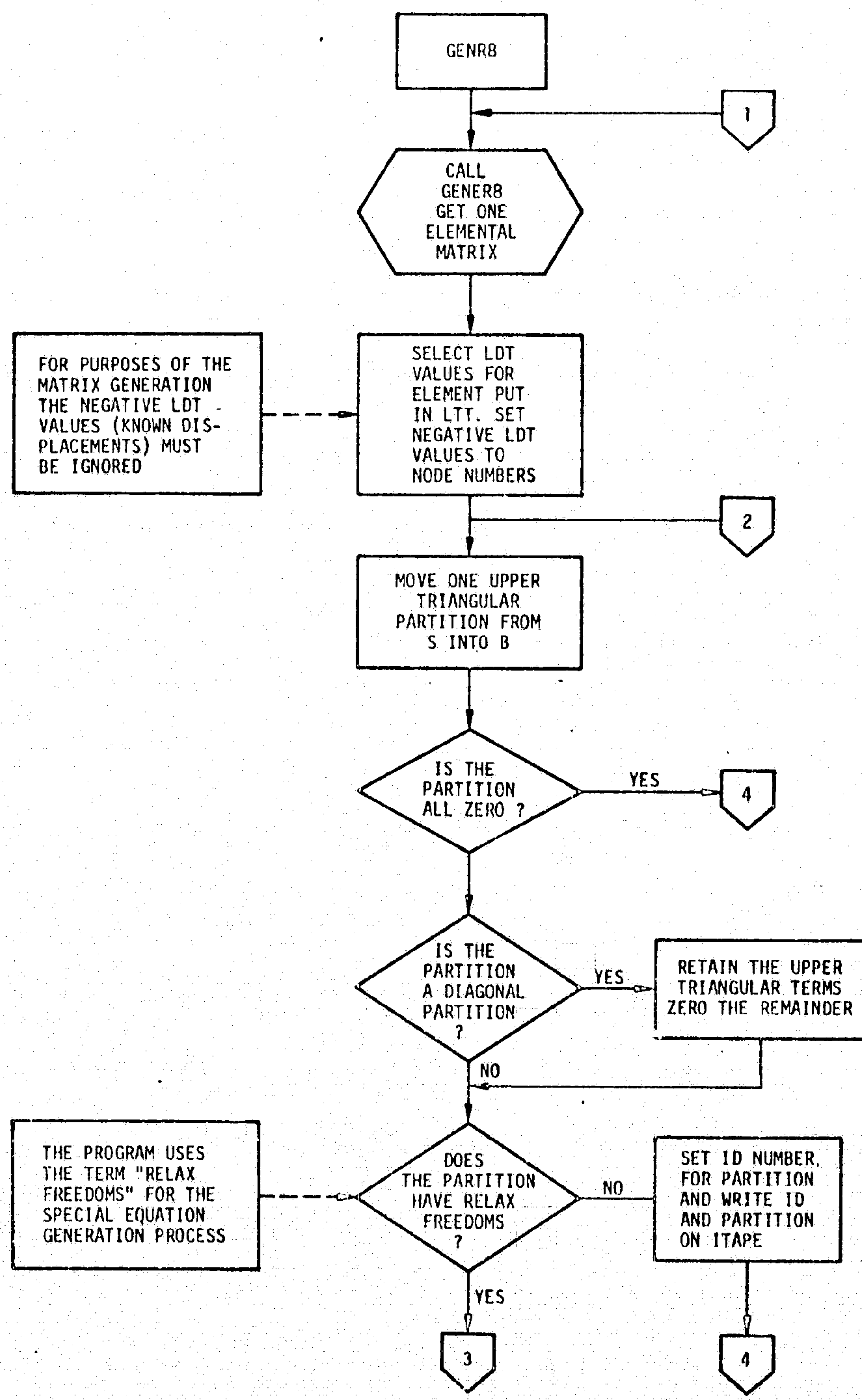
NODES = array of elemental nodes

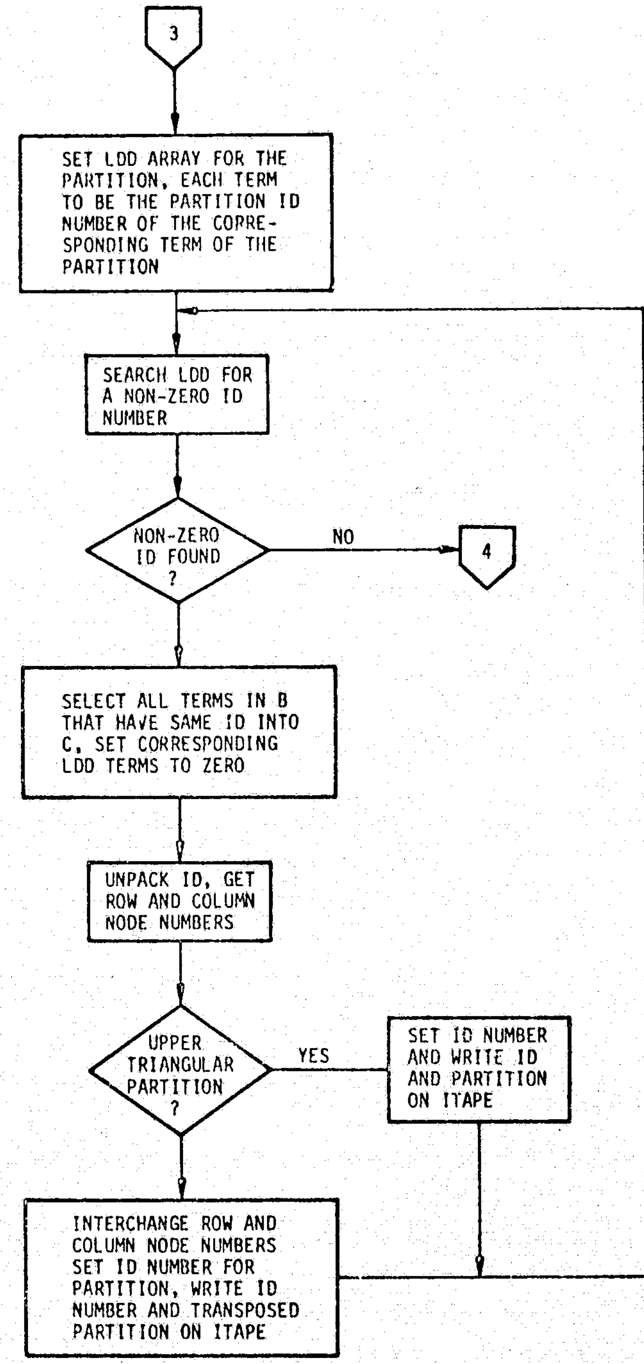
LDD = bookkeeping array

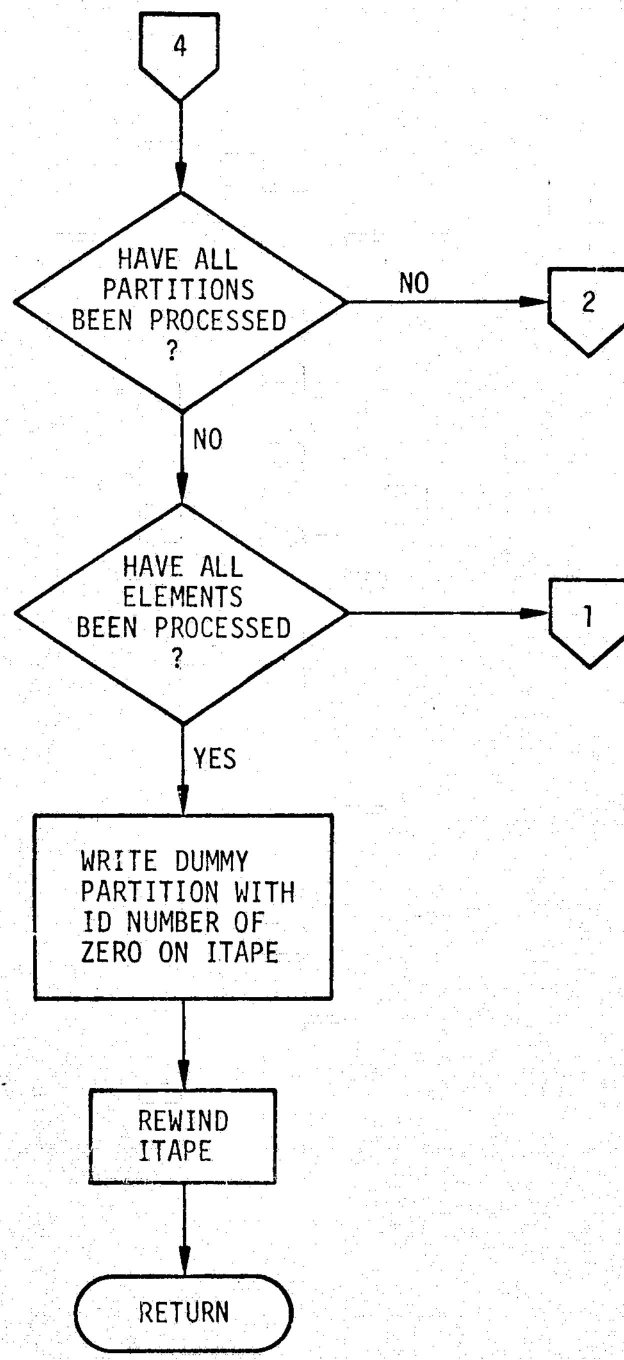
Array Dimensions

```
DOUBLE PRECISION S(NSN,NSN), B(NF,NF), C(NF,NF)
INTEGER NØDES(LEN), LDT(NN*NF), LTT(NSN), LDD(NF,NF)
```

where NN = number of nodes in problem


GENR8
1
CALL GENER8 GET ONE ELEMENTAL MATRIX
FOR PURPOSES OF THE MATRIX GENERATION THE NEGATIVE LDT VALUES (KNOWN DISPLACEMENTS) MUST BE IGNORED
SELECT LDT VALUES FOR ELEMENT PUT IN LTT. SET NEGATIVE LDT VALUES TO NODE NUMBERS
2
MOVE ONE UPPER TRIANGULAR PARTITION FROM S INTO B
IS THE PARTITION ALL ZERO ?
YES
4
IS THE PARTITION A DIAGONAL PARTITION ?
YES
RETAIN THE UPPER TRIANGULAR TERMS ZERO THE REMAINDER
NO
THE PROGRAM USES THE TERM "RELAX FREEDOMS" FOR THE SPECIAL EQUATION GENERATION PROCESS
DOES THE PARTITION HAVE RELAX FREEDOMS ?
NO
SET ID NUMBER, FOR PARTITION AND WRITE ID AND PARTITION ON ITAPE
YES
3
4


3
SET LDD ARRAY FOR THE PARTITION, EACH TERM TO BE THE PARTITION ID NUMBER OF THE CORRE-SPONDING TERM OF THE PARTITION
SEARCH LDD FOR A NON-ZERO ID NUMBER
NON-ZERO ID FOUND ?
NO
4
SELECT ALL TERMS IN B THAT HAVE SAME ID INTO C, SET CORRESPONDING LDD TERMS TO ZERO
UNPACK ID, GET ROW AND COLUMN NODE NUMBERS
UPPER TRIANGULAR PARTITION ?
YES
SET ID NUMBER AND WRITE ID AND PARTITION ON ITAPE
INTERCHANGE ROW AND COLUMN NODE NUMBERS SET ID NUMBER FOR PARTITION, WRITE ID NUMBER AND TRANSPOSED PARTITION ON ITAPE


4
HAVE ALL PARTITIONS BEEN PROCESSED ?
NO
2
NO
HAVE ALL ELEMENTS BEEN PROCESSED ?
1
YES
WRITE DUMMY PARTITION WITH ID NUMBER OF ZERO ON ITAPE
REWIND ITAPE
RETURN

6.3.2 (continued)

Subroutine MERSOR

Calling sequence

CALL MERSOR (NTAPE,KTAPE,LTAPE,NARG,MAX,DATA,TEMP,NDATA,LIST)

where

NTAPE = FORTRAN logical unit number of the input partitions to be merged and the output merged sorted partitions

KTAPE = FORTRAN logical unit numbers of a unit to be used as scratch

LTAPE = FORTRAN logical unit number of a unit to be used as scratch

NARG = number of words in one partition

MAX = number of partitions that can be kept incore

DATA = array for incore partition storage

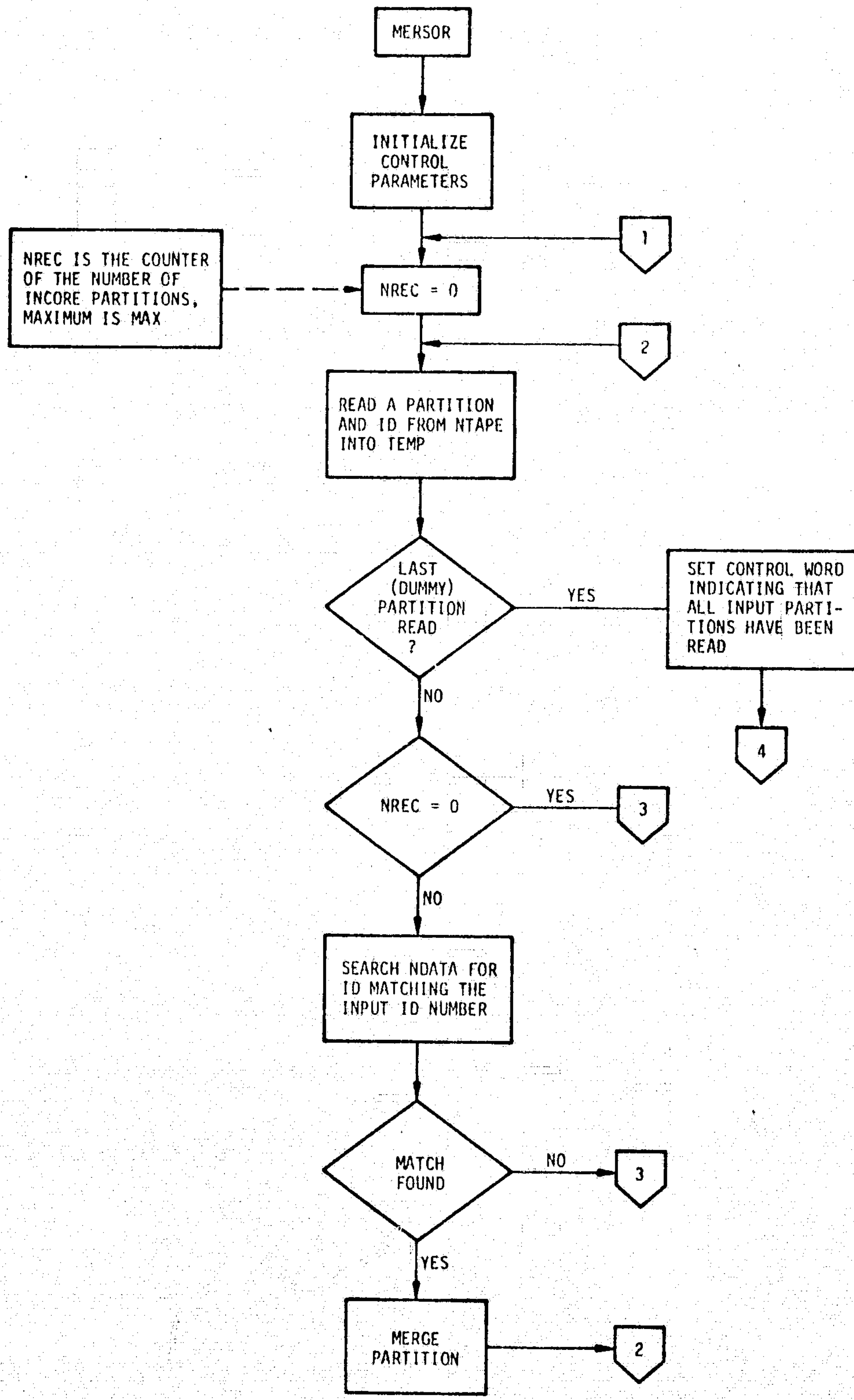
TEMP = array for one partition

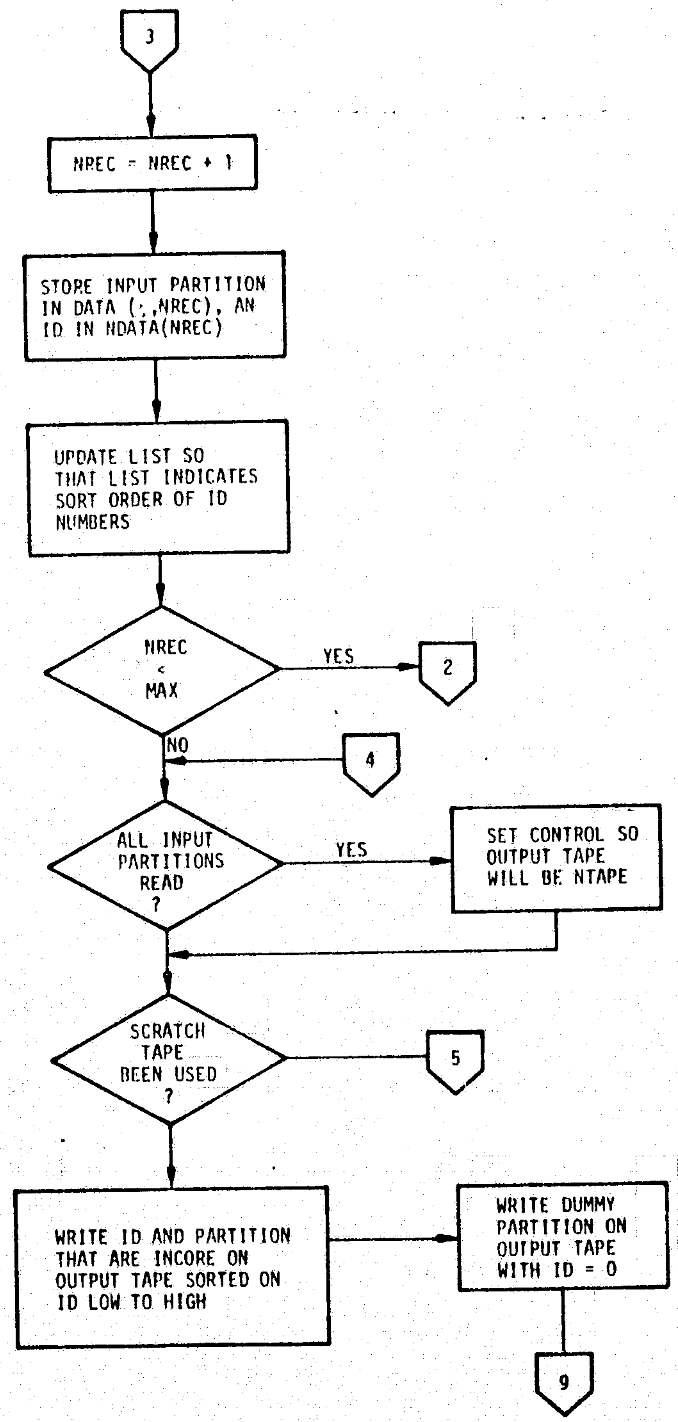
NDATA = array for incore partition ID numbers

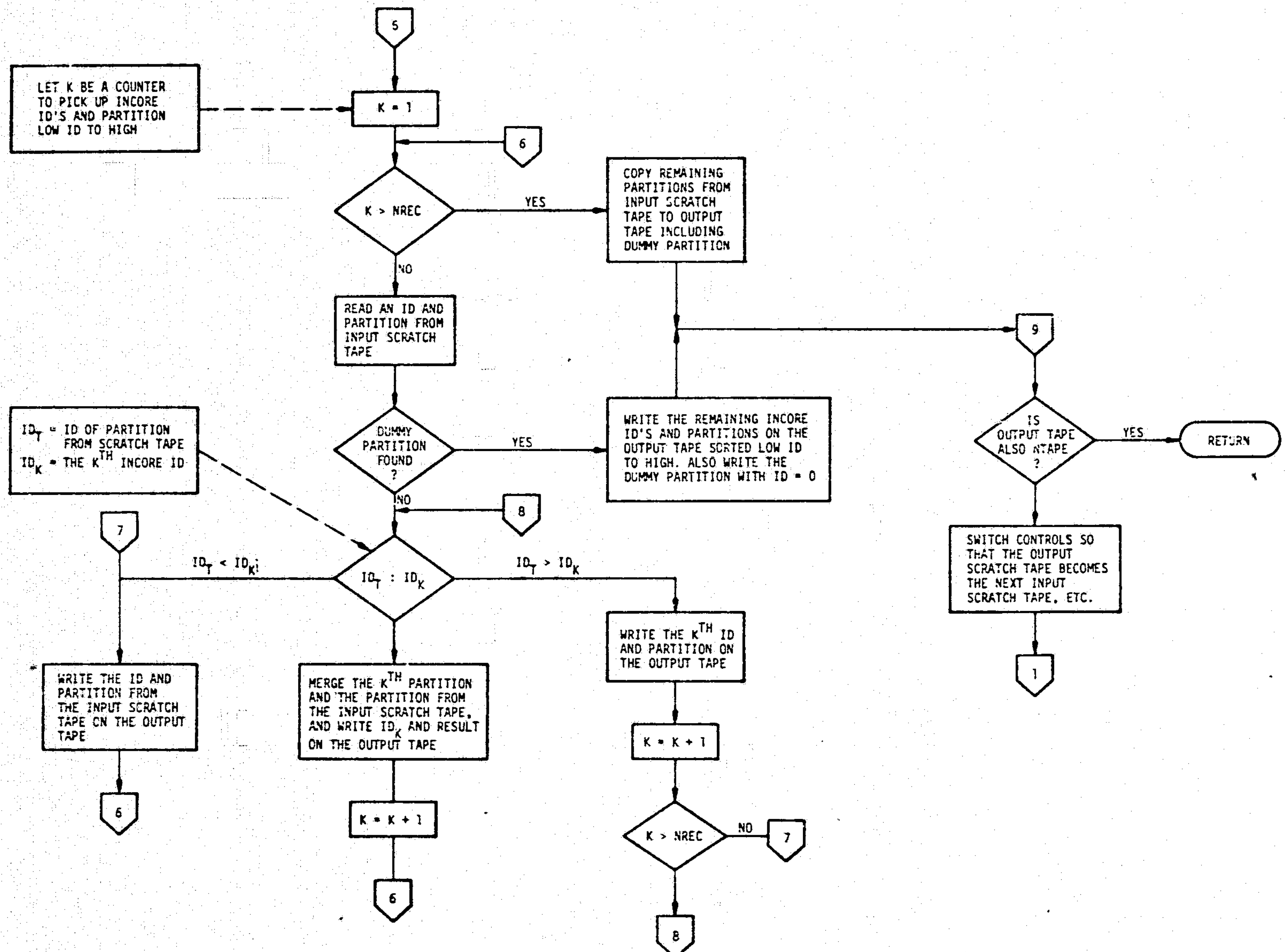
LIST = sorting array for ID numbers

Array Dimensions

```
DOUBLE PRECISION DATA(NARG,MAX), TEMP(NARG)
INTEGER NDATA(MAX), LIST(MAX)
```


MERSOR
INITIALIZE CONTROL PARAMETERS
1
NREC IS THE COUNTER OF THE NUMBER OF INCORE PARTITIONS, MAXIMUM IS MAX
NREC = 0
2
READ A PARTITION AND ID FROM NTAPE INTO TEMP
LAST (DUMMY) PARTITION READ ?
YES
SET CONTROL WORD INDICATING THAT ALL INPUT PARTITIONS HAVE BEEN READ
4
NO
NREC = 0
YES
3
NO
SEARCH NDATA FOR ID MATCHING THE INPUT ID NUMBER
MATCH FOUND
NO
3
YES
MERGE PARTITION
2


3
NREC = NREC + 1
STORE INPUT PARTITION IN DATA (·,NREC), AN ID IN NDATA(NREC)
UPDATE LIST SO THAT LIST INDICATES SORT ORDER OF ID NUMBERS
NREC < MAX
YES
2
NO
4
ALL INPUT PARTITIONS READ ?
YES
SET CONTROL SO OUTPUT TAPE WILL BE NTAPE
SCRATCH TAPE BEEN USED ?
5
WRITE ID AND PARTITION THAT ARE INCORE ON OUTPUT TAPE SORTED ON ID LOW TO HIGH
WRITE DUMMY PARTITION ON OUTPUT TAPE WITH ID = 0
9


5
LET K BE A COUNTER TO PICK UP INCORE ID'S AND PARTITION LOW ID TO HIGH
K = 1
6
K > NREC
YES
COPY REMAINING PARTITIONS FROM INPUT SCRATCH TAPE TO OUTPUT TAPE INCLUDING DUMMY PARTITION
NO
READ AN ID AND PARTITION FROM INPUT SCRATCH TAPE
9
DUMMY PARTITION FOUND ?
YES
WRITE THE REMAINING INCORE ID'S AND PARTITIONS ON THE OUTPUT TAPE SORTED LOW ID TO HIGH. ALSO WRITE THE DUMMY PARTITION WITH ID = 0
IS OUTPUT TAPE ALSO NTAPE ?
YES
RETURN
ID_T = ID OF PARTITION FROM SCRATCH TAPE
ID_K = THE K^{TH} INCORE ID
NO
8
7
$ID_T < ID_K$
$ID_T : ID_K$
$ID_T > ID_K$
SWITCH CONTROLS SO THAT THE OUTPUT SCRATCH TAPE BECOMES THE NEXT INPUT SCRATCH TAPE, ETC.
WRITE THE K^{TH} ID AND PARTITION ON THE OUTPUT TAPE
1
WRITE THE ID AND PARTITION FROM THE INPUT SCRATCH TAPE ON THE OUTPUT TAPE
MERGE THE K^{TH} PARTITION AND THE PARTITION FROM THE INPUT SCRATCH TAPE, AND WRITE ID_K AND RESULT ON THE OUTPUT TAPE
K = K + 1
6
K = K + 1
K > NREC
NO
7
6
8

6.3.2 (continued)

Subroutine DUMMY

CALL DUMMY (ITAPE,MBW,NFN,TEMP,MA)

where

ITAPE = FORTRAN logical unit number of the unit containing the merged ordered matrix

MBW = maximum bandwidth

NFN = number of words in one partition

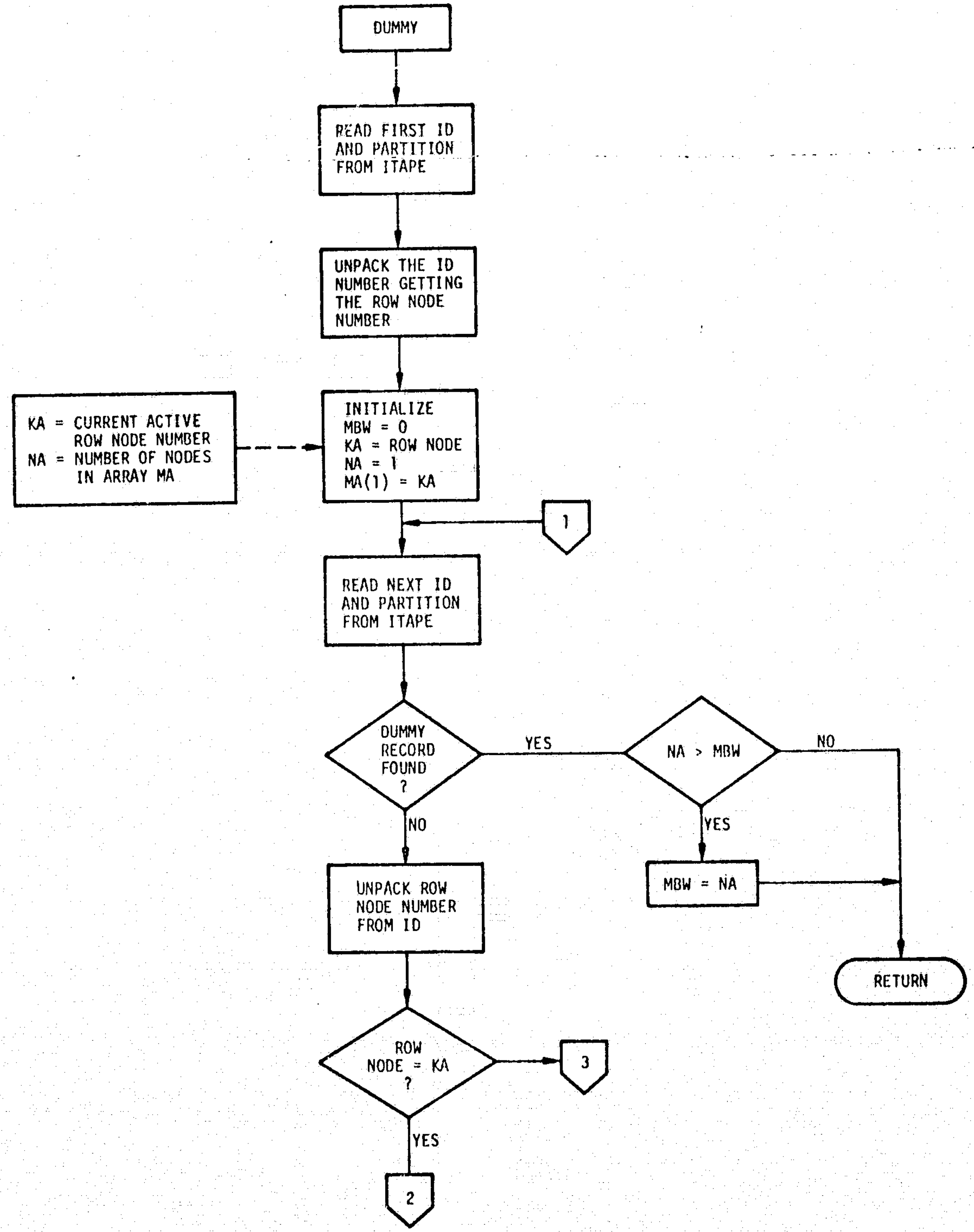
TEMP = array for input partition

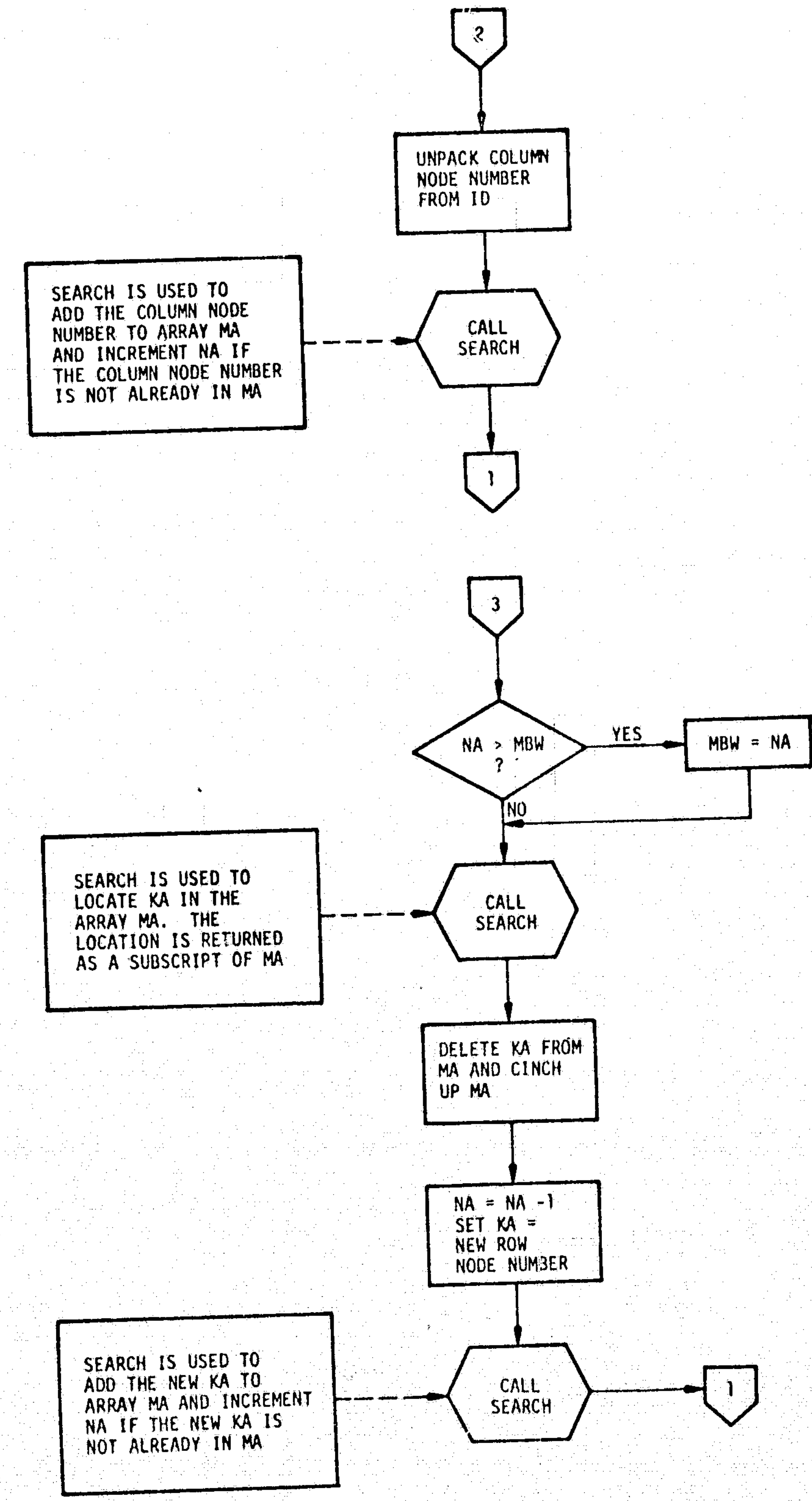
MA = array of active node numbers

Array Dimensions

```
DOUBLE PRECISION TEMP(NFN)
INTEGER MS(NN)
```

where NN = number of nodes in problem


DUMMY
READ FIRST ID AND PARTITION FROM ITAPE
UNPACK THE ID NUMBER GETTING THE ROW NODE NUMBER
KA = CURRENT ACTIVE ROW NODE NUMBER
NA = NUMBER OF NODES IN ARRAY MA
INITIALIZE
MBW = 0
KA = ROW NODE
NA = 1
MA(1) = KA
1
READ NEXT ID AND PARTITION FROM ITAPE
DUMMY RECORD FOUND ?
YES
NA > MBW
NO
YES
MBW = NA
RETURN
NO
UNPACK ROW NODE NUMBER FROM ID
ROW NODE = KA ?
3
YES
2


2
UNPACK COLUMN NODE NUMBER FROM ID
SEARCH IS USED TO ADD THE COLUMN NODE NUMBER TO ARRAY MA AND INCREMENT NA IF THE COLUMN NODE NUMBER IS NOT ALREADY IN MA
CALL SEARCH
1
3
NA > MBW ?
YES
MBW = NA
NO
SEARCH IS USED TO LOCATE KA IN THE ARRAY MA. THE LOCATION IS RETURNED AS A SUBSCRIPT OF MA
CALL SEARCH
DELETE KA FROM MA AND CINCH UP MA
NA = NA -1 SET KA = NEW ROW NODE NUMBER
SEARCH IS USED TO ADD THE NEW KA TO ARRAY MA AND INCREMENT NA IF THE NEW KA IS NOT ALREADY IN MA
CALL SEARCH
1

6.3.2 (continued)

Subroutine SEARCH

Calling Sequence

CALL SEARCH (N,K,MA,NA)

where

N is the node number to be located or added to array MA.

K is the location (subscript) of node N in array MA.

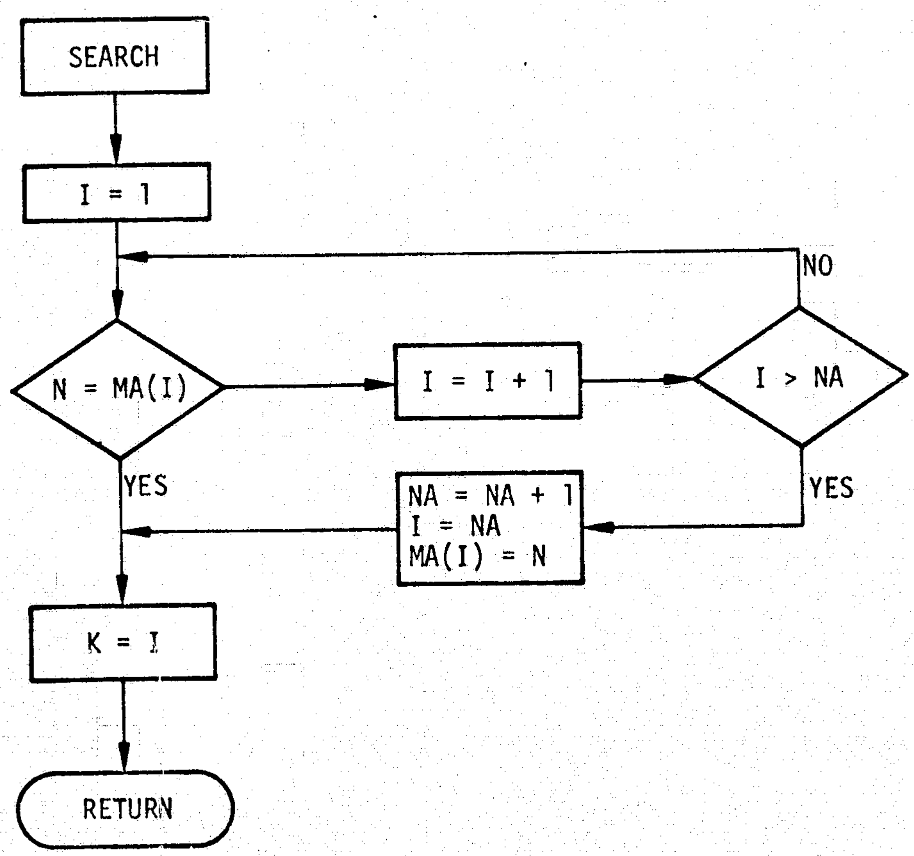
MA is an array of nodes.

NA is the number of nodes in array MA.

Array Dimensions

INTEGER MA (NN)

where NN is the number of nodes in the problem



6.3.3 DECOMPOSITION ROUTINES

The general flow of the Decomposition routine is shown on Figure 6.3-6.

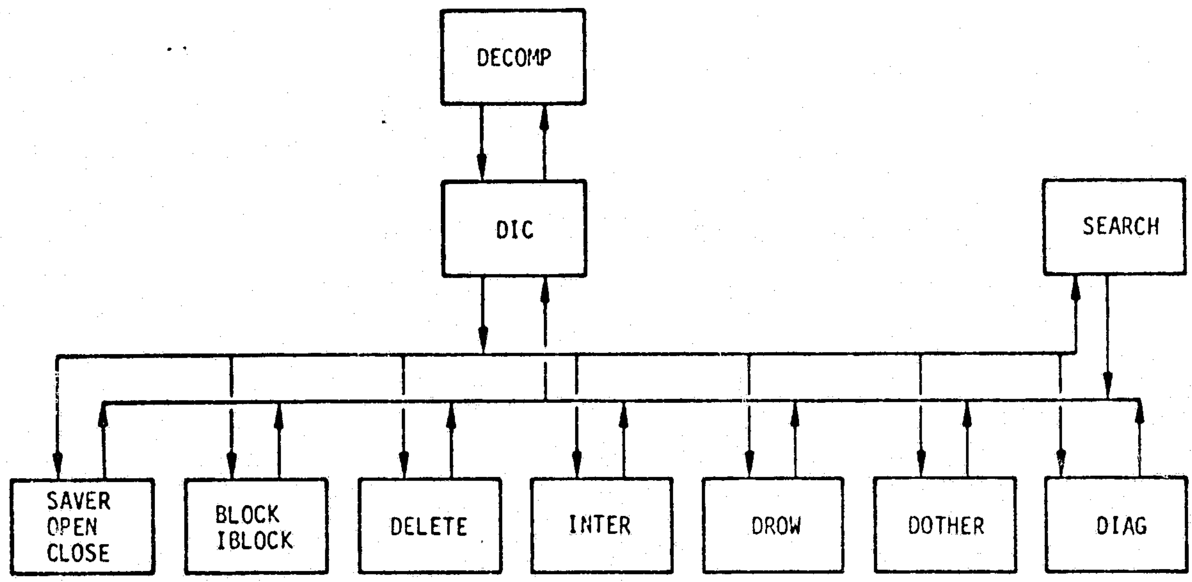


FIGURE 6.3-6: DECOMPOSITION FLOW

DECOMP is the "MAIN" entry subroutine called by the user. Its main functions are to 1) compute storage assignments for the arrays needed in the routines; and 2) to test the storage requirements against the amounts of storage available, and halt the run when insufficient storage is available. Storage assignments are in common block /JLB/ and are passed via the calling sequence arguments.

DIC is the incore decomposition routine performing the actual decomposition.

SAVER/OPEN/CLOSE is one routine with three entry points. OPEN initializes the decomposition tape and writes the header record. SAVER moves the nodal record data into the data records and writes the data records on the decomposition tape when they are full. CLOSE writes the trailer record on the decomposition tape.

6.3.3 (continued)

BLOCK/IBLOCK is one routine with two entry points. Both operate on the partition storage usage bookkeeping array. IBLOCK initializes the array. BLOCK determines the next available partition storage block to be used.

DELETE operates on the partition storage bookkeeping array, releasing partition storage blocks for later use.

INTER operates on the partition storage bookkeeping array and the active node array, interchanging the items in these arrays so that the node of the row being decomposed is the last node in the active node bookkeeping array.

DIAG does the internal decomposition of the diagonal partition of the row being decomposed.

DROW does the decomposition of the off-diagonal partitions in the row being decomposed.

DOTHER does the decomposition of the row into the partitions not in the row.

STORAGE AND BOOKKEEPING CONCEPTS

The key concept used during decomposition is that the only partitions required incore are those that are active in the decomposition for the row being decomposed. The number of partitions requiring storage is

$$NBLKS = (\tfrac{1}{2})\ (MBW)(MBW+1) \tag{6.3-1}$$

where MBW is the maximum bandwidth.

6.3.3 (continued)

Three bookkeeping arrays are needed, an active node array, a partition row/column ID versus partition storage block number array, and a partition storage use array. In practice the storage for the last two bookkeeping arrays are combined by packing the values.

The partition storage blocks are a double dimensioned array, the first dimension for the terms in the partition, the second the partition storage block number (in the range 1 to NBLKS). There is a one to one correspondence between the partition storage block second index and the storage use bookkeeping array. The storage use array is simply a flag indicating whether or not its corresponding partition storage block is used by an active partition.

The active node bookkeeping array is used in conjunction with the partition row/column ID bookkeeping array to locate a specific partition in the partition storage blocks. This is done by thinking of the partition row/column ID bookkeeping array as an upper triantular matrix as shown in Figure 6.3-7.

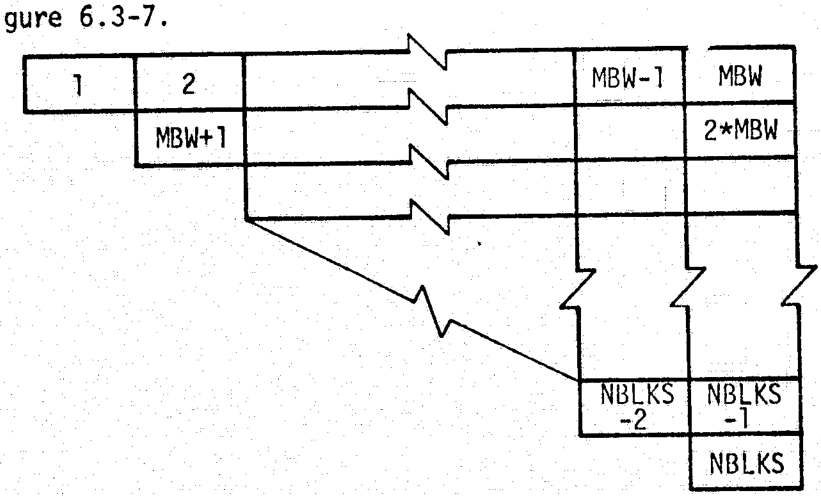


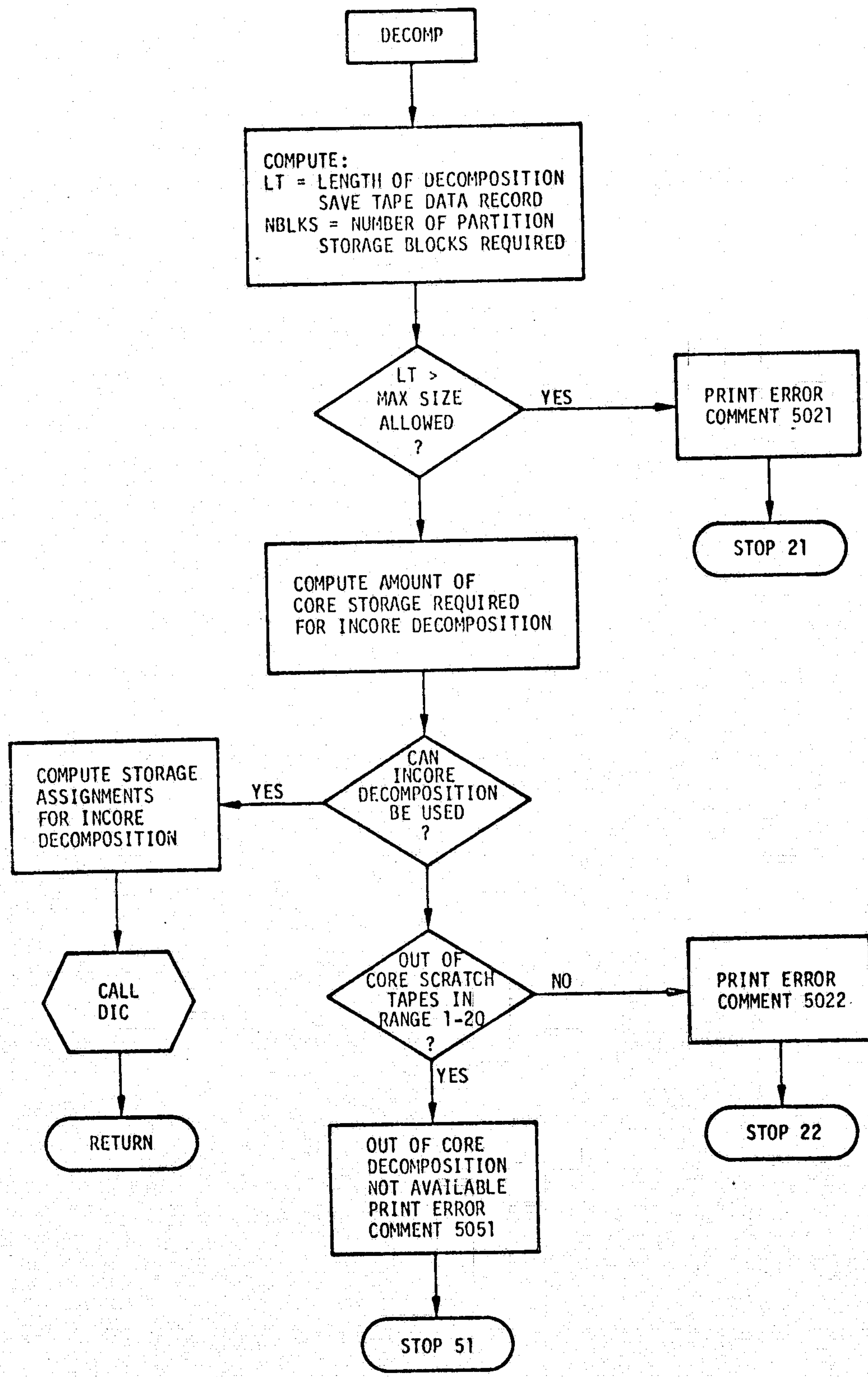
FIGURE 6.3-7: PARTITION ROW/COLUMN ID BOOKKEEPING ARRAY

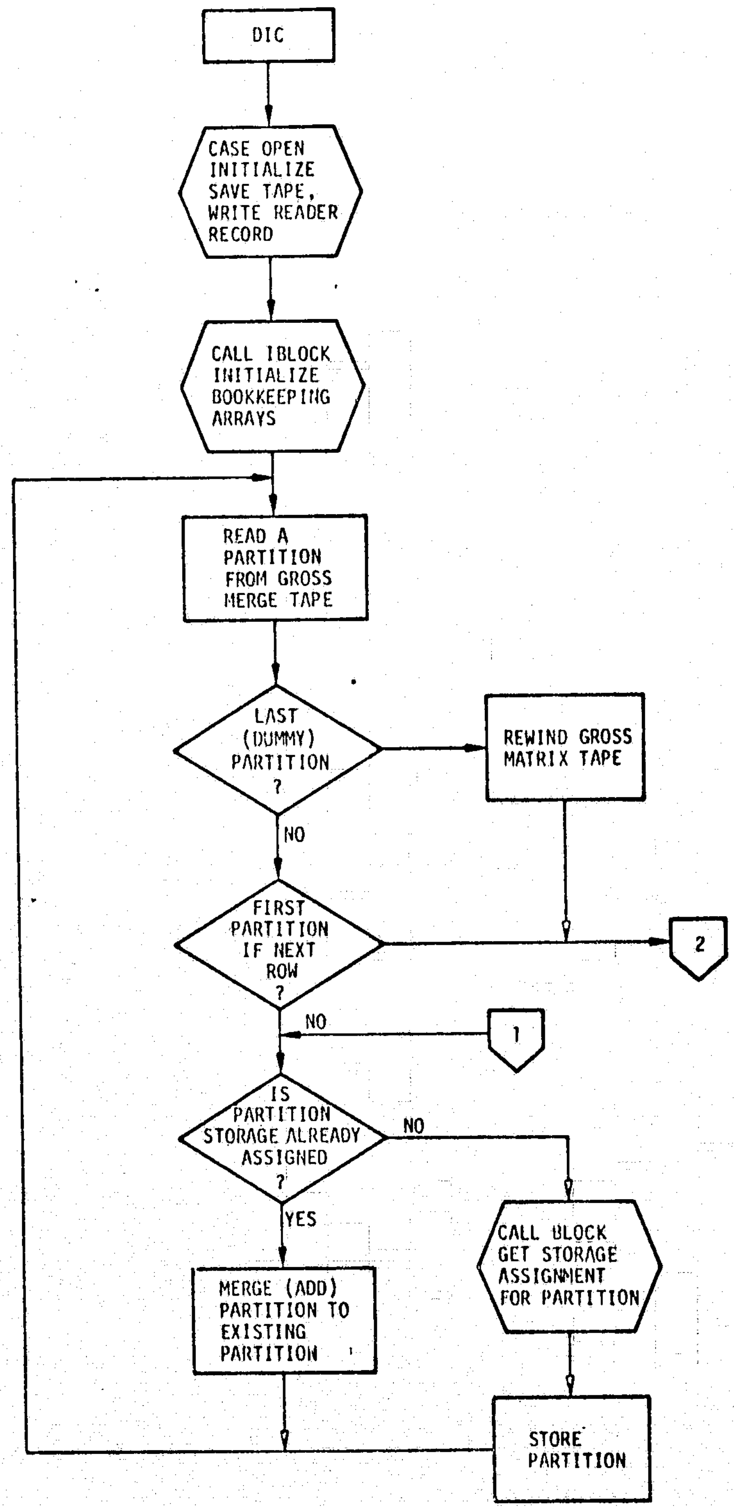
6.3.3 (continued)

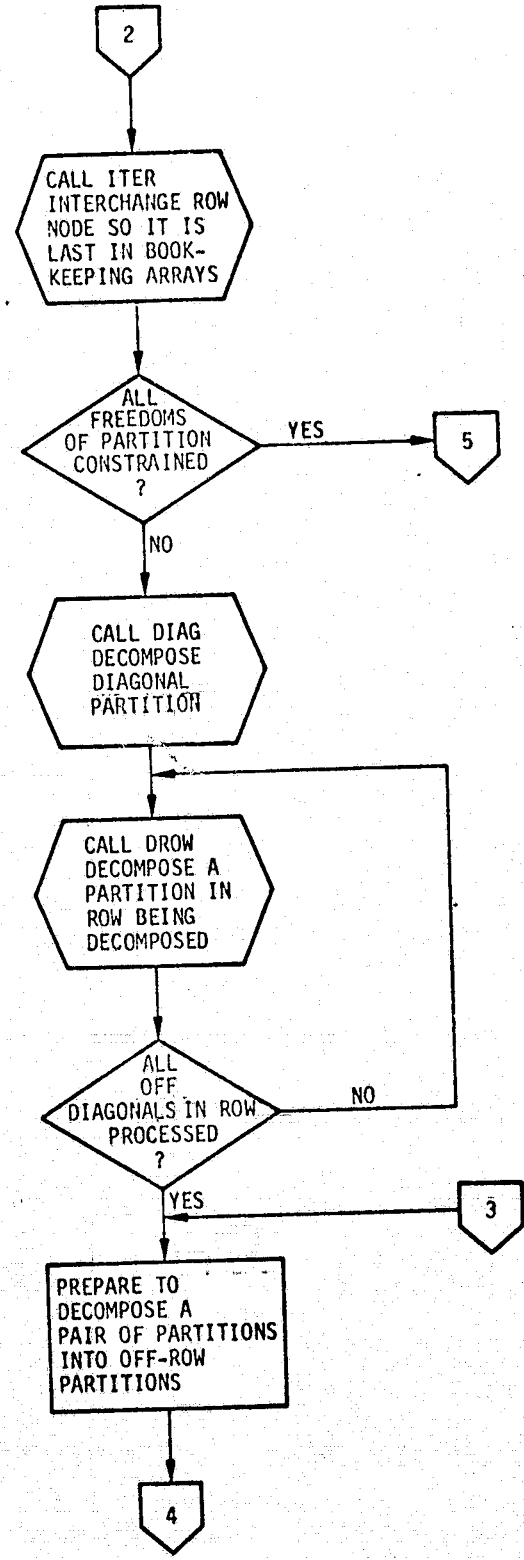
There is a one to one correspondence between the active nodes bookkeeping array and the rows and columns in this upper triangular matrix. To locate a specific partition, say partition (I,J) (row node I, column node J). First, locate the nodes I and J in the active node bookkeeping array (the node numbers are stored in the array), let i and j be their location (subscript) respectively. Second, if j is less than i ($j < i$), interchange j and i. Finally, the location (second index of the partition storage blocks) of the partition is stored in the partition row/column ID bookkeeping array at location k of that bookkeeping array,

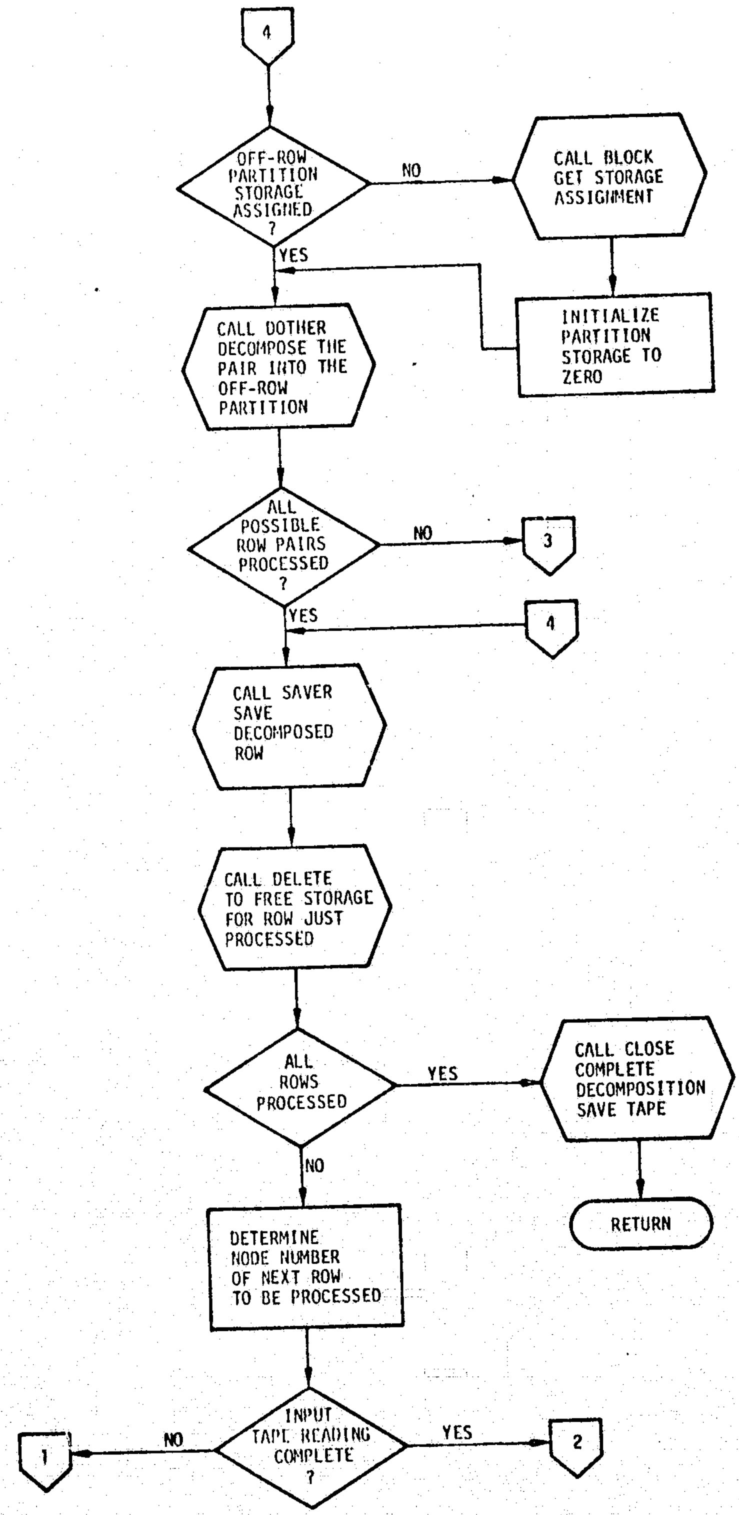
where

$$k = (\tfrac{1}{2}) * (i) * (2*MBW - i + 1) - MBW + j \qquad (6.3\text{-}2)$$

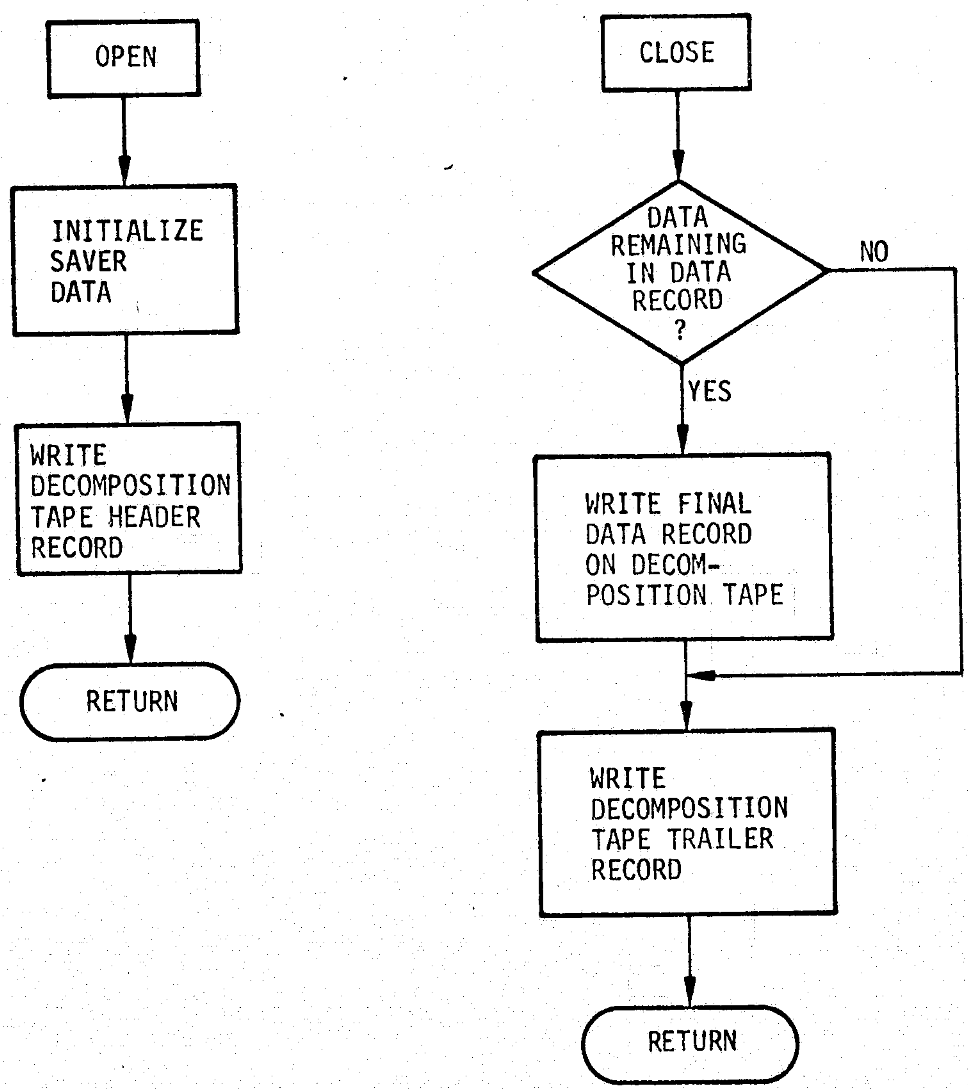

DECOMP
COMPUTE:
LT = LENGTH OF DECOMPOSITION SAVE TAPE DATA RECORD
NBLKS = NUMBER OF PARTITION STORAGE BLOCKS REQUIRED
LT > MAX SIZE ALLOWED ?
YES
PRINT ERROR COMMENT 5021
STOP 21
COMPUTE AMOUNT OF CORE STORAGE REQUIRED FOR INCORE DECOMPOSITION
CAN INCORE DECOMPOSITION BE USED ?
YES
COMPUTE STORAGE ASSIGNMENTS FOR INCORE DECOMPOSITION
CALL DIC
RETURN
OUT OF CORE SCRATCH TAPES IN RANGE 1-20 ?
NO
PRINT ERROR COMMENT 5022
STOP 22
YES
OUT OF CORE DECOMPOSITION NOT AVAILABLE PRINT ERROR COMMENT 5051
STOP 51

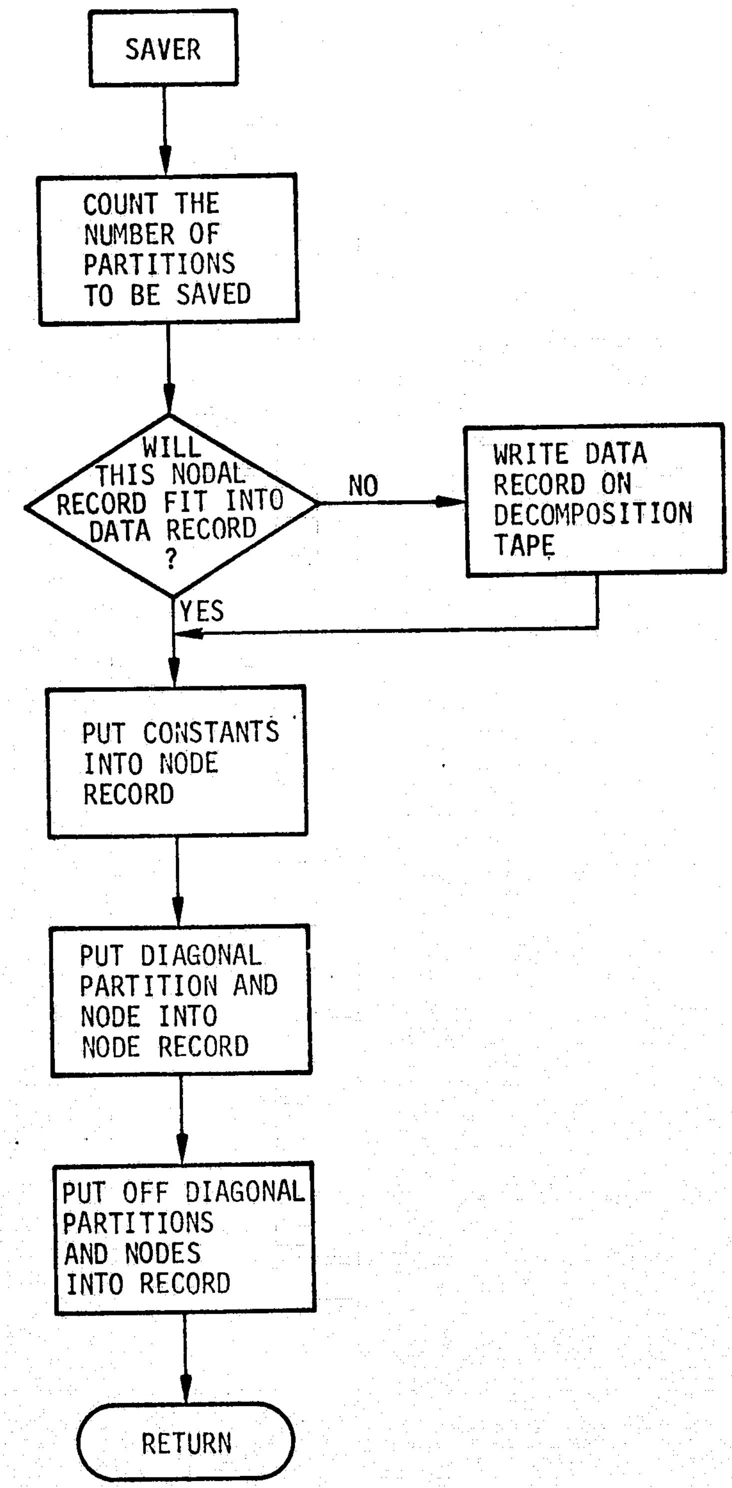

DIC
CASE OPEN INITIALIZE SAVE TAPE, WRITE READER RECORD
CALL IBLOCK INITIALIZE BOOKKEEPING ARRAYS
READ A PARTITION FROM GROSS MERGE TAPE
LAST (DUMMY) PARTITION ?
REWIND GROSS MATRIX TAPE
NO
FIRST PARTITION IF NEXT ROW ?
2
NO
1
IS PARTITION STORAGE ALREADY ASSIGNED ?
NO
YES
CALL BLOCK GET STORAGE ASSIGNMENT FOR PARTITION
MERGE (ADD) PARTITION TO EXISTING PARTITION
STORE PARTITION

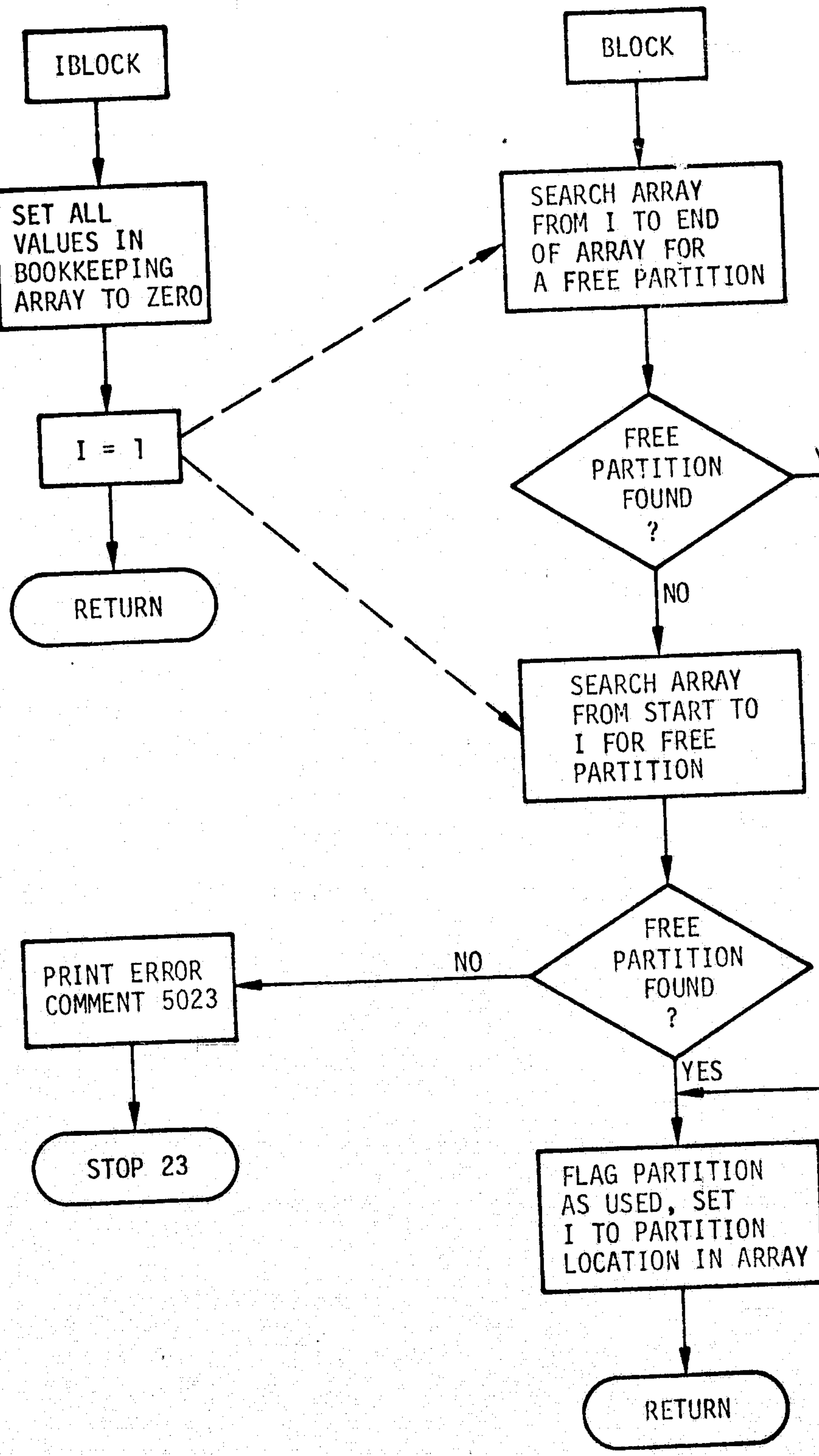

2
CALL ITER
INTERCHANGE ROW
NODE SO IT IS
LAST IN BOOK-
KEEPING ARRAYS
ALL
FREEDOMS
OF PARTITION
CONSTRAINED
?
YES
5
NO
CALL DIAG
DECOMPOSE
DIAGONAL
PARTITION
CALL DROW
DECOMPOSE A
PARTITION IN
ROW BEING
DECOMPOSED
ALL
OFF
DIAGONALS IN ROW
PROCESSED
?
NO
YES
3
PREPARE TO
DECOMPOSE A
PAIR OF PARTITIONS
INTO OFF-ROW
PARTITIONS
4

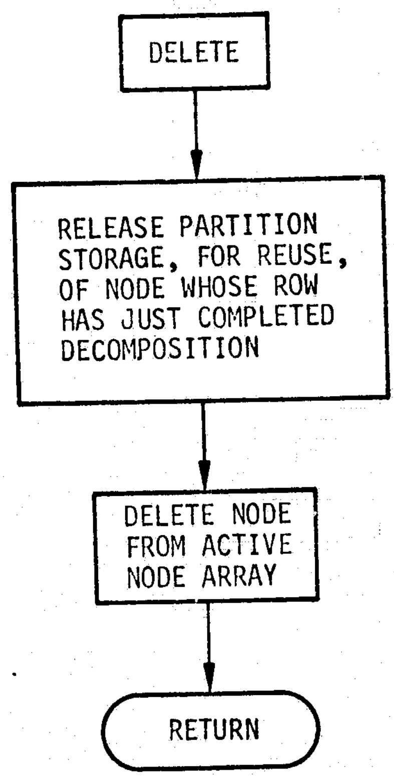


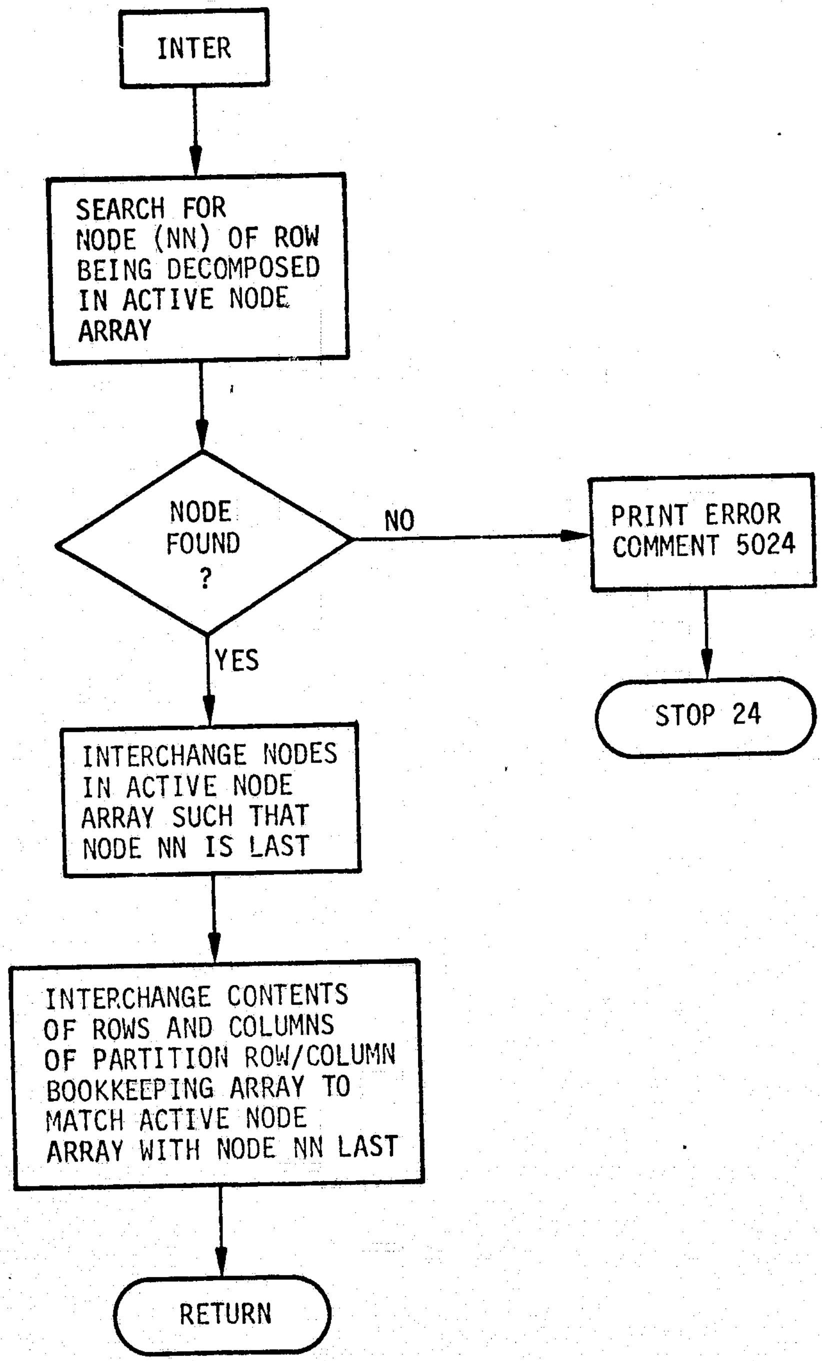
4
OFF-ROW PARTITION STORAGE ASSIGNED ?
NO
CALL BLOCK GET STORAGE ASSIGNMENT
YES
INITIALIZE PARTITION STORAGE TO ZERO
CALL DOTHER DECOMPOSE THE PAIR INTO THE OFF-ROW PARTITION
ALL POSSIBLE ROW PAIRS PROCESSED ?
NO
3
YES
4
CALL SAVER SAVE DECOMPOSED ROW
CALL DELETE TO FREE STORAGE FOR ROW JUST PROCESSED
ALL ROWS PROCESSED
YES
CALL CLOSE COMPLETE DECOMPOSITION SAVE TAPE
RETURN
NO
DETERMINE NODE NUMBER OF NEXT ROW TO BE PROCESSED
INPUT TAPE READING COMPLETE ?
NO
1
YES
2

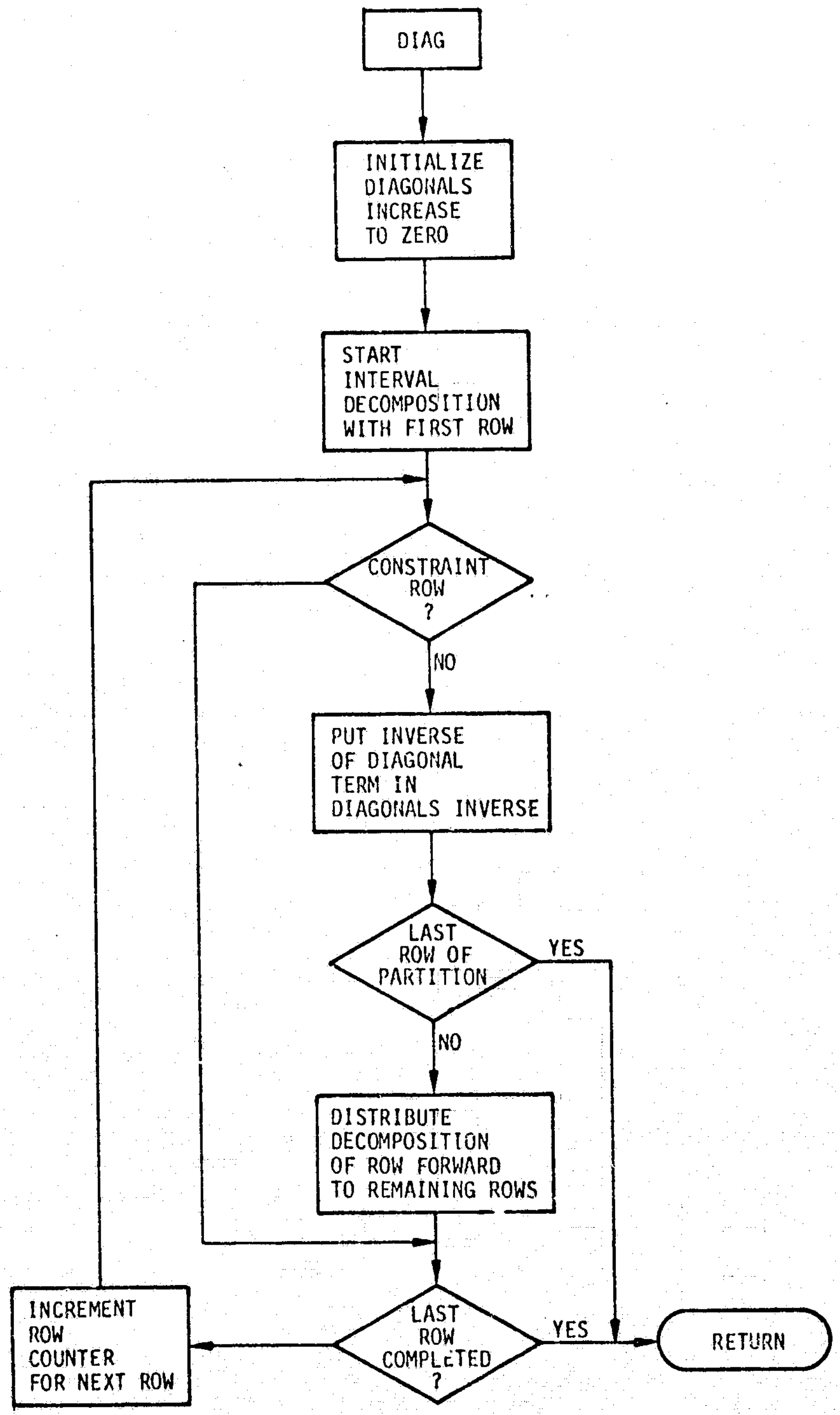

OPEN
INITIALIZE SAVER DATA
WRITE DECOMPOSITION TAPE HEADER RECORD
RETURN
CLOSE
DATA REMAINING IN DATA RECORD ?
NO
YES
WRITE FINAL DATA RECORD ON DECOM-POSITION TAPE
WRITE DECOMPOSITION TAPE TRAILER RECORD
RETURN

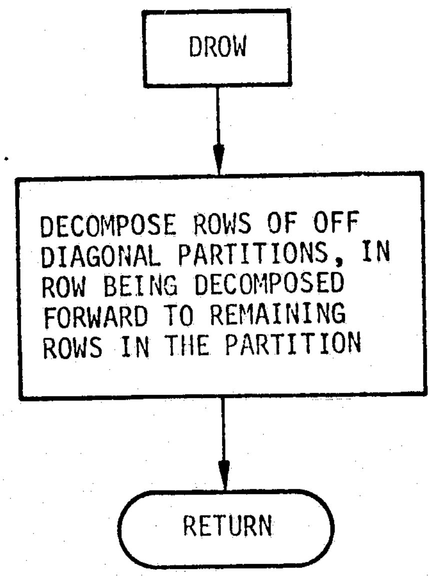

SAVER
COUNT THE NUMBER OF PARTITIONS TO BE SAVED
WILL THIS NODAL RECORD FIT INTO DATA RECORD ?
NO
WRITE DATA RECORD ON DECOMPOSITION TAPE
YES
PUT CONSTANTS INTO NODE RECORD
PUT DIAGONAL PARTITION AND NODE INTO NODE RECORD
PUT OFF DIAGONAL PARTITIONS AND NODES INTO RECORD
RETURN


IBLOCK
SET ALL VALUES IN BOOKKEEPING ARRAY TO ZERO
I = 1
RETURN
BLOCK
SEARCH ARRAY FROM I TO END OF ARRAY FOR A FREE PARTITION
FREE PARTITION FOUND ?
YES
NO
SEARCH ARRAY FROM START TO I FOR FREE PARTITION
FREE PARTITION FOUND ?
NO
YES
PRINT ERROR COMMENT 5023
STOP 23
FLAG PARTITION AS USED, SET I TO PARTITION LOCATION IN ARRAY
RETURN


DELETE
RELEASE PARTITION STORAGE, FOR REUSE, OF NODE WHOSE ROW HAS JUST COMPLETED DECOMPOSITION
DELETE NODE FROM ACTIVE NODE ARRAY
RETURN


INTER
SEARCH FOR NODE (NN) OF ROW BEING DECOMPOSED IN ACTIVE NODE ARRAY
NODE FOUND ?
NO
PRINT ERROR COMMENT 5024
STOP 24
YES
INTERCHANGE NODES IN ACTIVE NODE ARRAY SUCH THAT NODE NN IS LAST
INTERCHANGE CONTENTS OF ROWS AND COLUMNS OF PARTITION ROW/COLUMN BOOKKEEPING ARRAY TO MATCH ACTIVE NODE ARRAY WITH NODE NN LAST
RETURN


DIAG
INITIALIZE DIAGONALS INCREASE TO ZERO
START INTERVAL DECOMPOSITION WITH FIRST ROW
CONSTRAINT ROW ?
NO
PUT INVERSE OF DIAGONAL TERM IN DIAGONALS INVERSE
LAST ROW OF PARTITION
YES
NO
DISTRIBUTE DECOMPOSITION OF ROW FORWARD TO REMAINING ROWS
INCREMENT ROW COUNTER FOR NEXT ROW
LAST ROW COMPLETED ?
YES
RETURN


DROW
DECOMPOSE ROWS OF OFF DIAGONAL PARTITIONS, IN ROW BEING DECOMPOSED FORWARD TO REMAINING ROWS IN THE PARTITION
RETURN

$[A] = [C]^T[B][D]$

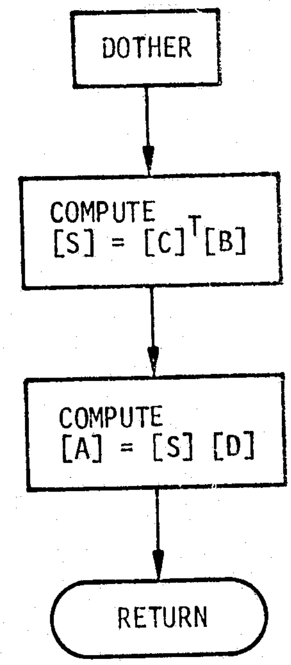
WHERE

[C] AND [D] ARE OFF DIAGONAL PARTITIONS IN THE ROW BEING DECOMPOSED

[B] IS THE INVERSE OF THE DIAGONAL OF THE DIAGONAL PARTITION IN THE ROW BEING DECOMPOSED

[A] IS THE RECEIVING PARTITION

[S] IS A SCRATCH PARTITION



6.3.4 FORWARD/BACKWARD SUBSTITUTION ROUTINES

The general flow of the Forward/Backward Substitution routines is shown in Figure 6.3-8

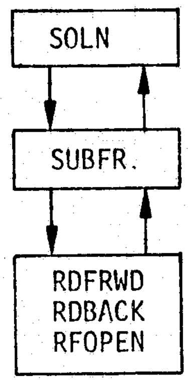
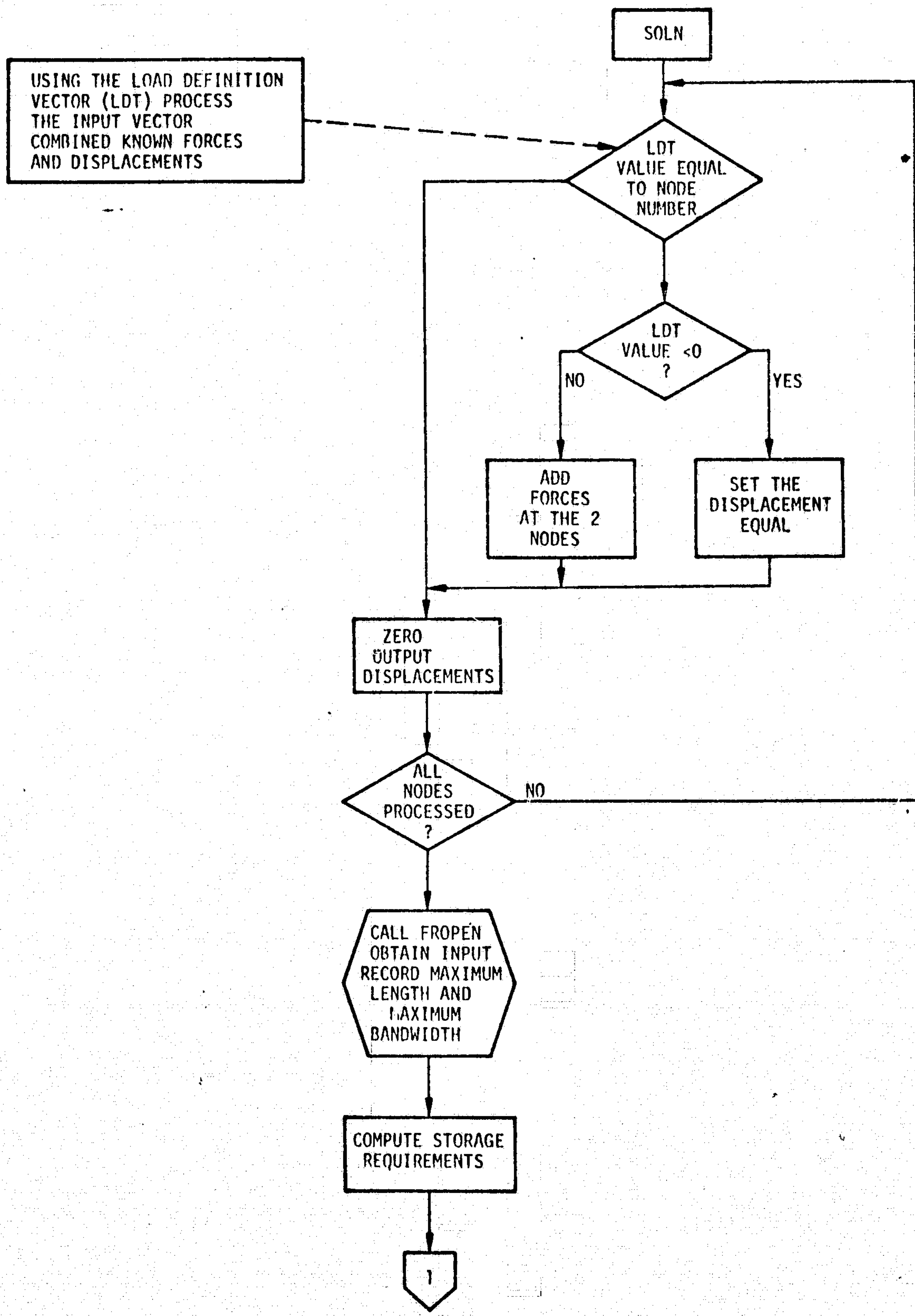
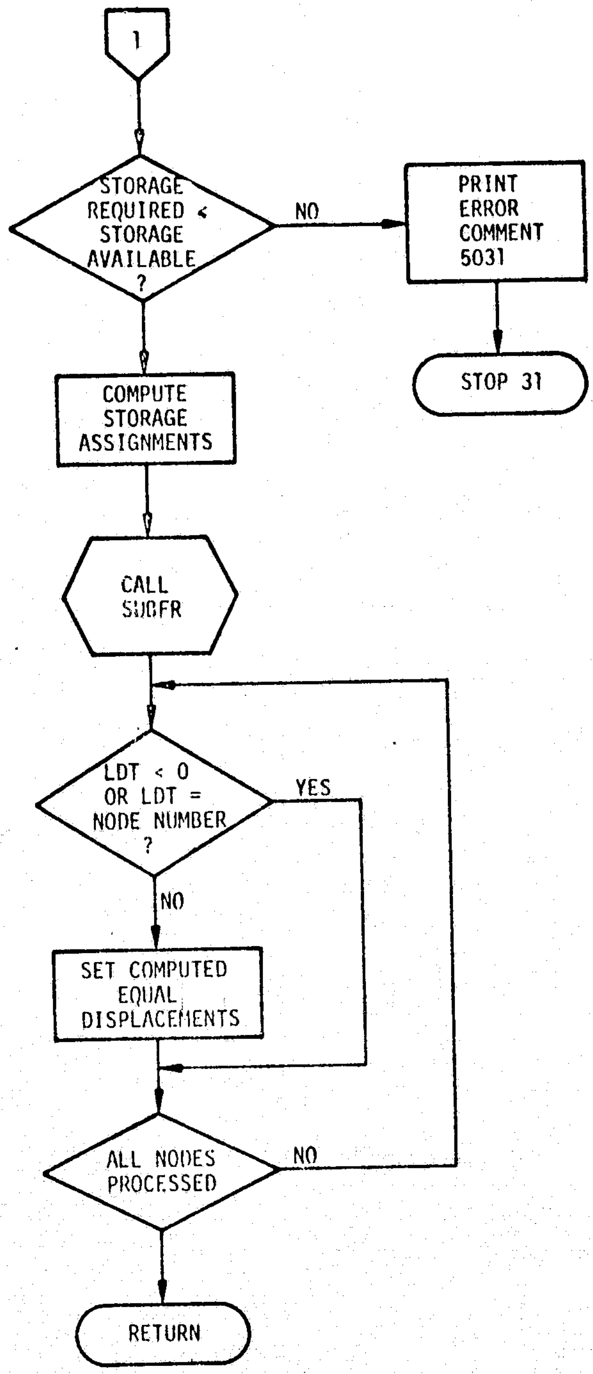


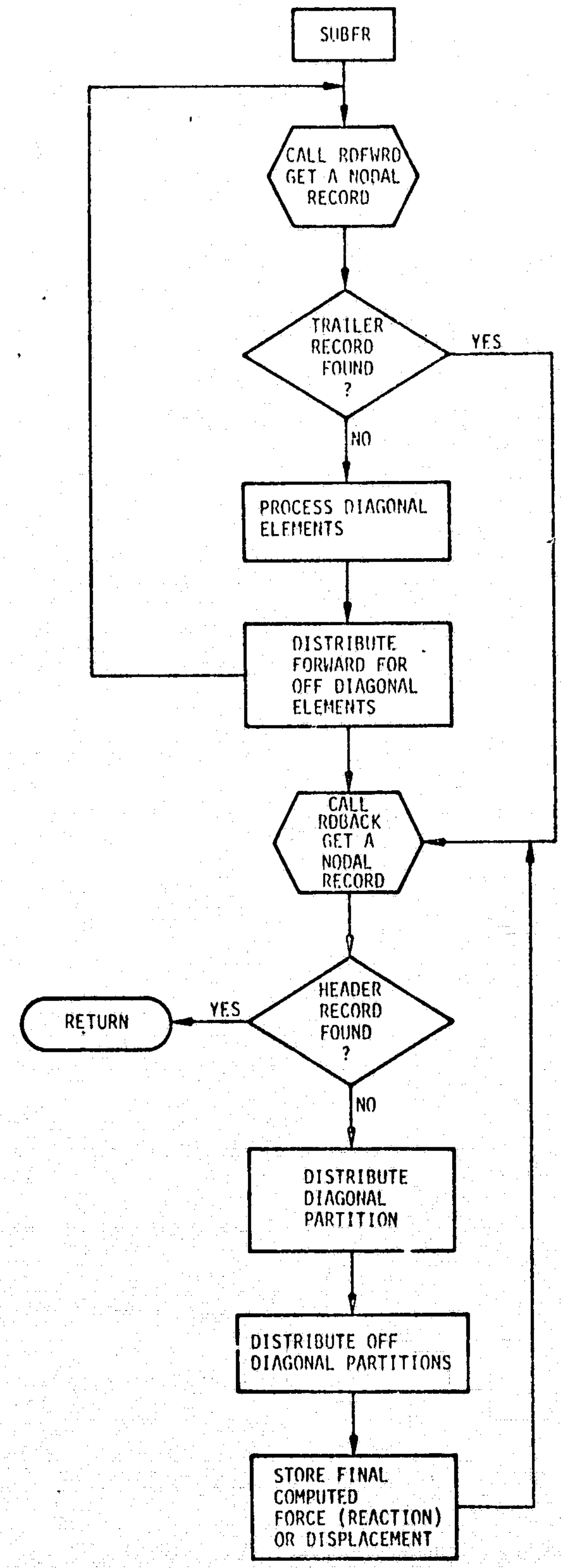
Figure 6.3-8 Substitution Flow

SOLN is the "MAIN" entry subroutine called by the user. Its main functions are to 1) set prescribed equal known (input) displacements, and add known (input) forces on computed equal displacement nodes, 2) to compute storage assignments for the arrays needed in the process, and 3) after substitution is completed, set computed equal displacements. Storage assignments are in common block/JLB/' and are passed via the calling sequence arguements.

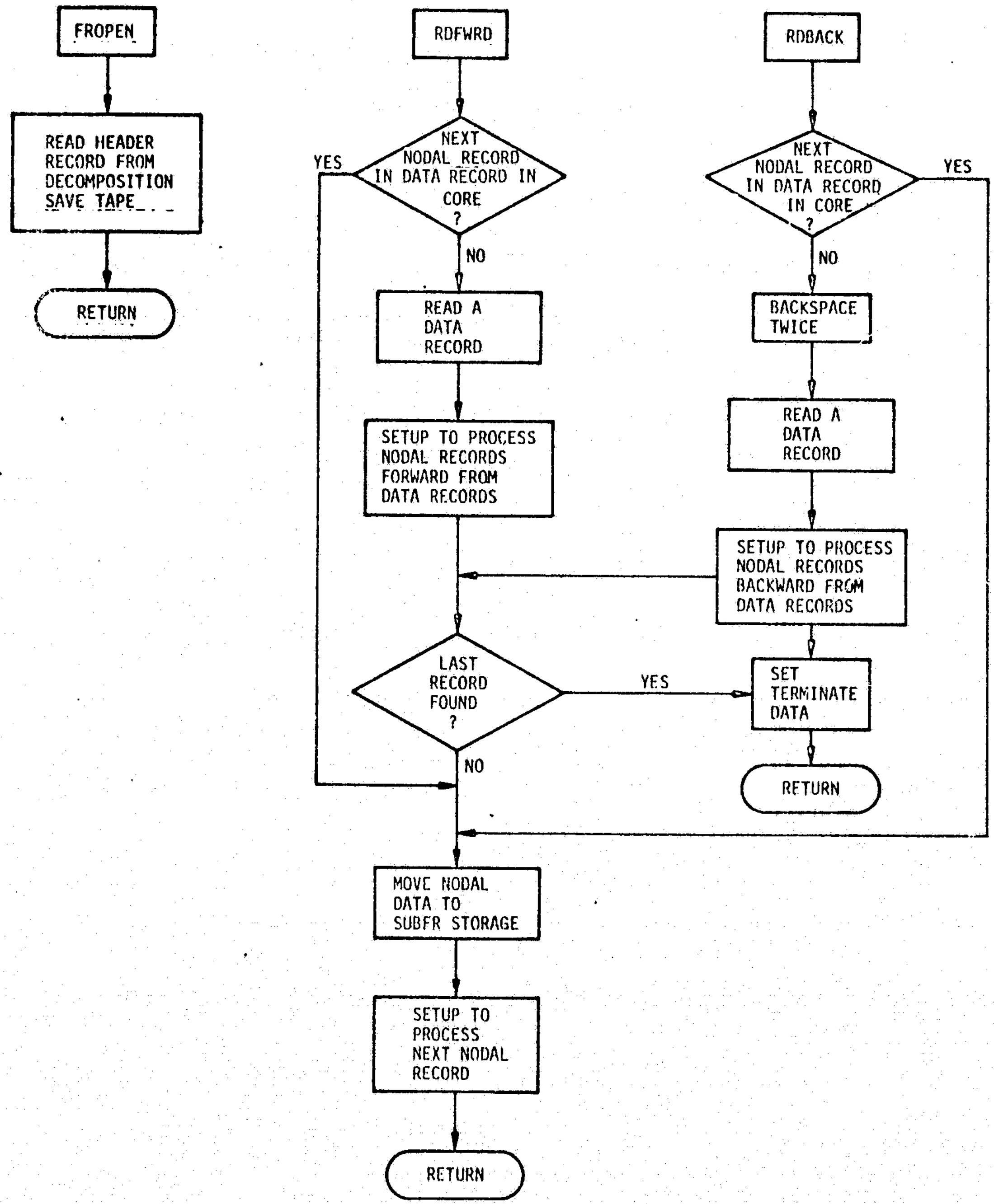
SUBFR is the forward/backward substitution routine performing the actual substitution RDFWRD/RDBACK/FROPEN is one routine with three entry points. All three read the decomposition save tape. FROPEN reads the header record, returning the maximum bandwidth and length of a data record to SOLN for use in computing storage assignments RDFWRD and RDBACK each read a node record. RDFWRD reads the tape for forward substitution and RDBACK reads the tape backwards for backsubstitution.


SOLN
USING THE LOAD DEFINITION VECTOR (LDT) PROCESS THE INPUT VECTOR COMBINED KNOWN FORCES AND DISPLACEMENTS
LDT VALUE EQUAL TO NODE NUMBER
LDT VALUE <0 ?
NO
YES
ADD FORCES AT THE 2 NODES
SET THE DISPLACEMENT EQUAL
ZERO OUTPUT DISPLACEMENTS
ALL NODES PROCESSED ?
NO
CALL FROPEN OBTAIN INPUT RECORD MAXIMUM LENGTH AND MAXIMUM BANDWIDTH
COMPUTE STORAGE REQUIREMENTS
1

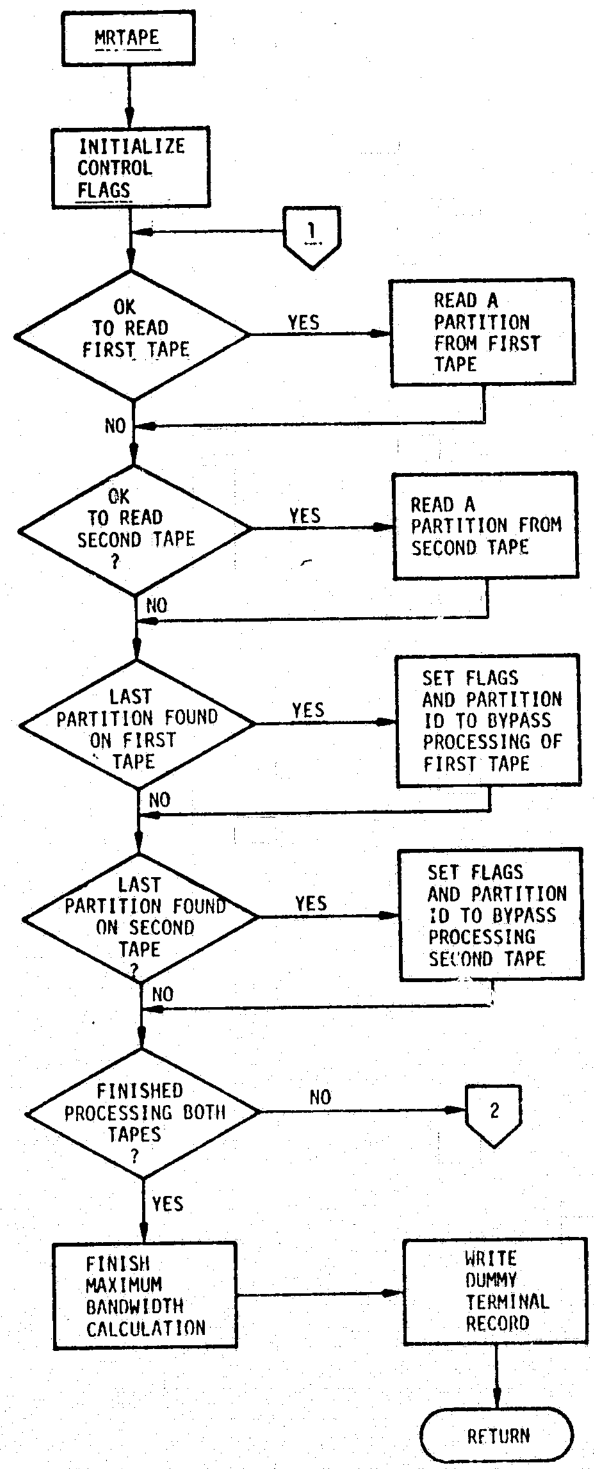

1
STORAGE REQUIRED ≤ STORAGE AVAILABLE ?
NO
PRINT ERROR COMMENT 5031
STOP 31
COMPUTE STORAGE ASSIGNMENTS
CALL SUBFR
LDT < 0 OR LDT = NODE NUMBER ?
YES
NO
SET COMPUTED EQUAL DISPLACEMENTS
ALL NODES PROCESSED
NO
RETURN

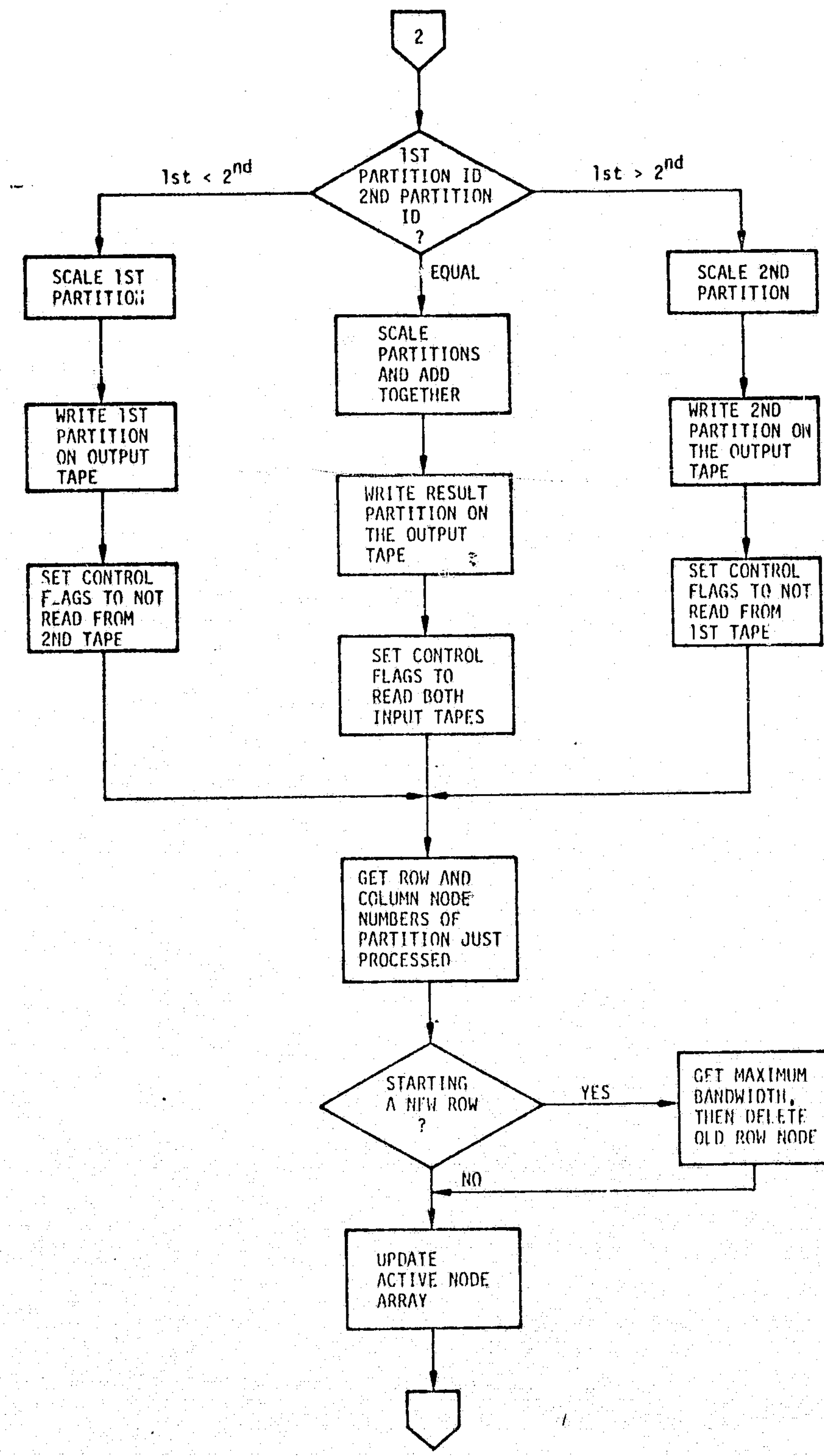


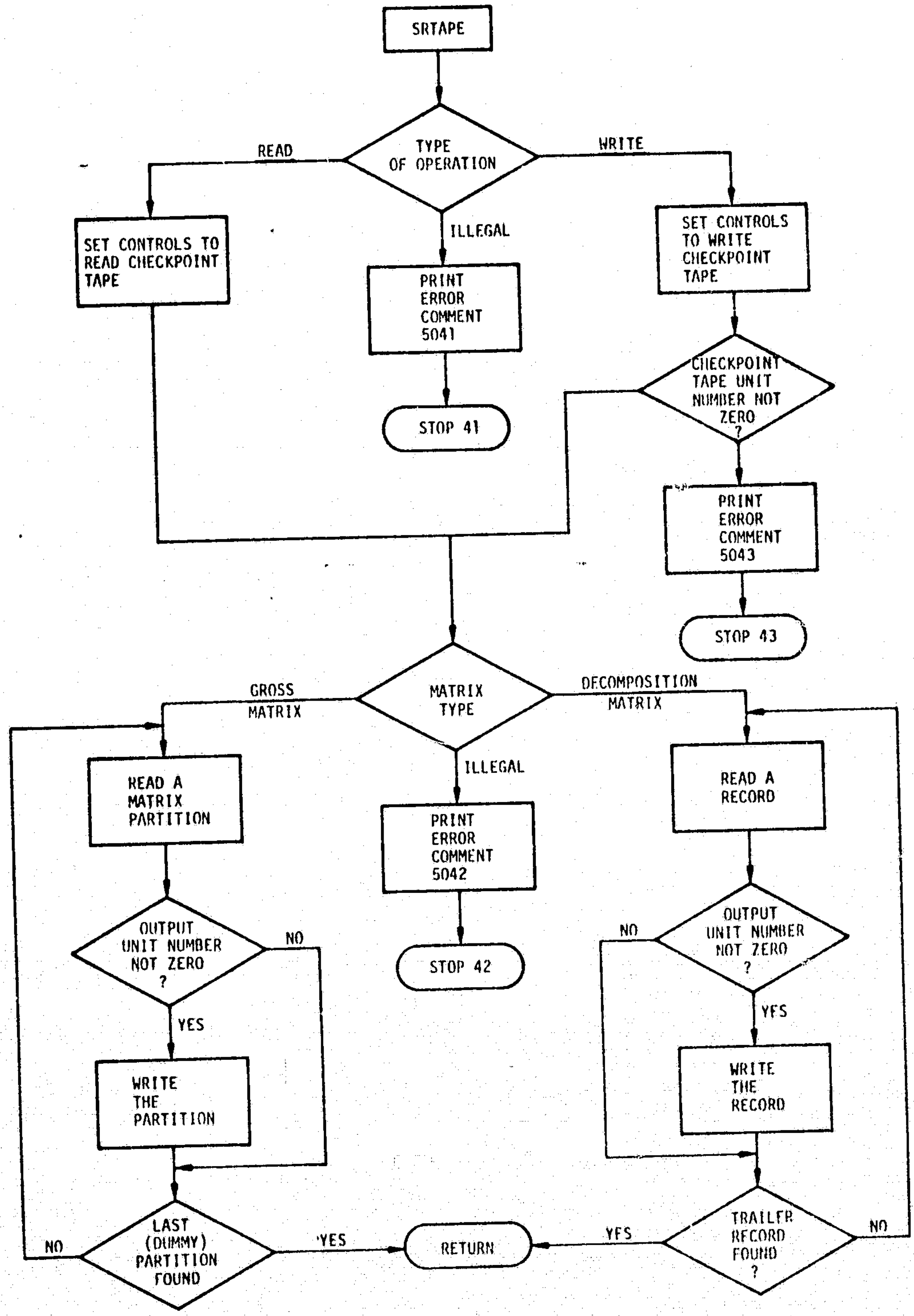
SUBFR
CALL RDFWRD GET A NODAL RECORD
TRAILER RECORD FOUND ?
YES
NO
PROCESS DIAGONAL ELEMENTS
DISTRIBUTE FORWARD FOR OFF DIAGONAL ELEMENTS
CALL RDBACK GET A NODAL RECORD
HEADER RECORD FOUND ?
YES
RETURN
NO
DISTRIBUTE DIAGONAL PARTITION
DISTRIBUTE OFF DIAGONAL PARTITIONS
STORE FINAL COMPUTED FORCE (REACTION) OR DISPLACEMENT


FROPEN
READ HEADER RECORD FROM DECOMPOSITION SAVE TAPE
RETURN
RDFWRD
NEXT NODAL RECORD IN DATA RECORD IN CORE ?
YES
NO
READ A DATA RECORD
SETUP TO PROCESS NODAL RECORDS FORWARD FROM DATA RECORDS
LAST RECORD FOUND ?
YES
NO
MOVE NODAL DATA TO SUBFR STORAGE
SETUP TO PROCESS NEXT NODAL RECORD
RETURN
RDBACK
NEXT NODAL RECORD IN DATA RECORD IN CORE ?
YES
NO
BACKSPACE TWICE
READ A DATA RECORD
SETUP TO PROCESS NODAL RECORDS BACKWARD FROM DATA RECORDS
SET TERMINATE DATA
RETURN

6.3.5 SPECIAL ROUTINES




2
1ST PARTITION ID 2ND PARTITION ID ?
1st < 2nd
1st > 2nd
EQUAL
SCALE 1ST PARTITION
WRITE 1ST PARTITION ON OUTPUT TAPE
SET CONTROL FLAGS TO NOT READ FROM 2ND TAPE
SCALE PARTITIONS AND ADD TOGETHER
WRITE RESULT PARTITION ON THE OUTPUT TAPE
SET CONTROL FLAGS TO READ BOTH INPUT TAPES
SCALE 2ND PARTITION
WRITE 2ND PARTITION ON THE OUTPUT TAPE
SET CONTROL FLAGS TO NOT READ FROM 1ST TAPE
GET ROW AND COLUMN NODE NUMBERS OF PARTITION JUST PROCESSED
STARTING A NEW ROW ?
YES
GET MAXIMUM BANDWIDTH, THEN DELETE OLD ROW NODE
NO
UPDATE ACTIVE NODE ARRAY


SRTAPE
TYPE OF OPERATION
READ
WRITE
ILLEGAL
SET CONTROLS TO READ CHECKPOINT TAPE
PRINT ERROR COMMENT 5041
STOP 41
SET CONTROLS TO WRITE CHECKPOINT TAPE
CHECKPOINT TAPE UNIT NUMBER NOT ZERO ?
PRINT ERROR COMMENT 5043
STOP 43
MATRIX TYPE
GROSS MATRIX
DECOMPOSITION MATRIX
ILLEGAL
READ A MATRIX PARTITION
PRINT ERROR COMMENT 5042
STOP 42
READ A RECORD
OUTPUT UNIT NUMBER NOT ZERO ?
NO
YES
WRITE THE PARTITION
OUTPUT UNIT NUMBER NOT ZERO ?
NO
YES
WRITE THE RECORD
LAST (DUMMY) PARTITION FOUND
NO
YES
RETURN
YES
TRAILER RECORD FOUND ?
NO